YORK UNIVERSITY

TENTANDA VIA

YORK UNIVERSITY

APL CONFERENCE

HELD ON JANUARY 27/72

COMPUTER SERVICES

Further copies may be obtained by sending a purchase order or money order for $2.00 per copy to:

Miss Diane Huybers
Computer Services
Steacie Science Library, Room T106
York University
Downsview, Ont.

A G E N D A

8:45 - 9:00	REGISTRATION AND COFFEE	12:00 - 1:30	LUNCH
9:00 - 9:10	CHAIRMAN'S WELCOME Fred Simpkin	1:30 - 2:30	COMPUTER ASSISTED LEARNING Dr. P. Meincke
9:10 - 9:45	THE FUTURE OF APL IN EDUCATIONAL INSTITUTIONS Don Kellett	2:30 - 3:30	TEACHING MATHEMATICS WITH APL Dr. R. Roden
9:45 - 10:30	BUSINESS APPLICATIONS IN APL Mrs. Charlotte Seaberg	3:30 - 4:00	COFFEE
10:30 - 11:00	COFFEE	4:00 - 4:30	FILE I/O IN YORK APL Bob George
11:00 - 12:00	MULTI-TERMINAL SIMULATION EXPERIMENTS: A CASE IN REGIONAL PLANNING Dr. P. Medow	4:30 - 5:00	DISCUSSION PERIOD.

TABLE OF CONTENTS

THE FUTURE OF APL IN EDUCATIONAL INSTITUTIONS

BY

Mr. D. Kellett
York University

There is a tendency on the part of APL users to become extremely enthusiastic, almost to the point of talking like an evangelist. This trait is most noticeable when one is talking to other believers. Indeed, the proceedings of past APL conferences tend to give the impression of speakers lining up to testify like converts to a new religion. I am definitely enthusiastic about APL, as I hope most of you are. I will try to overcome this exuberance since I am sure we all can agree that one computing language will not satisfy everyone's needs, and that no system is perfect. What I plan to do today is share my thoughts about where APL could and should be going in the future. More particularly, I am interested in how APL can fit into the future computing needs of education. I hope to set the stage for the papers which will follow, and to generate lively discussion on the uses you have or plan for APL in your installations.

Naturally, the basic question one would ask is - does APL have a future in our educational computing needs? Your presence here today is an indication of the interest that there is in APL. In the last two years even IBM has recognized that the language exists, that it is of value to their customers and that it is a product that they can market. I.P. Sharpe Associates are proving that APL is commercially viable as a time-sharing language in the business environment. The most significant indication of the growing importance of APL here in Ontario is its use in Universities and Schools. Six Ontario Universities and Ryerson Polytechnical Institute currently support APL systems in their computing centres. Queens' University has a large number of APL users, in spite of the fact that there is no system on campus. Carlton and Laurentian Universities are evaluating or planning to implement systems in the near future. Of the fifteen Universities in this province probably ten will be using APL by the end of this year. This represents a significant user community. When you combine this with the availability of 1130 APL systems in high schools and the small but growing interest in purchasing APL time-sharing by community colleges, you can see that APL is more than a passing fad. Let's consider its future developments and the implications arising from them.

This increased use of APL has occurred on the strength of the language's character, and in spite of significant obstacles hindering its expansion. On the Hardware side, it is unfortunate that APL has been limited to the machinery of one manufacturer and in particular to a terminal device which is slow, noisy and expensive. It looks as if some relief is in sight for this situation in the near future. Texas Instruments plan to release their "silent" high speed hard-copy terminal with an APL character set, and Gandalf in Ottawa who market a cheap, portable CRT have informed us of plans to develop an APL display. As the number of APL users increases, terminal suppliers will be adapting and creating devices to support APL, giving us the same opportunities as those which are available to users of other time-sharing systems. Soon we should be able to acquire cheap, efficient and durable terminals that are a necessity for wide-spread student use of APL. (As a side note: I might mention that York APL has supported teletype-compatible devices since its inception. This convenient extension has allowed us to experiment with a variety of terminal hardware).

Other main-frame manufacturers are becoming interested in offering APL systems to their customers. Xerox Data Systems plans to release APL later this year, and Control Data Corporation has an established APL project group. I take this to be a sign of APL's increasing stature in the computing world.

To date, there have been three implementations of APL generally available: The basic IBM systems for 1130's or /360's: I.P. Sharpe's APL PLUS, a greatly enhansed version of the basic IBM/360 APL; and, York APL another implementation originally developed to run on a system not dedicated to time-sharing. Each has slightly different characteristics but all have the same common difficulties.

The main problem is the fact that APL is an interpretive system-an effective tool for writing and debugging-but questionable on the grounds of efficiency for production systems or large-scale number crunching. It is possible to write very efficient programs by taking advantage of the interpreter and the language, however this requires highly skilled programmers. On the whole there is a definite need for a compiler or semi-compiler for APL programs. Future hardware developments, that is, low cost high speed core or hard-wired systems, might tend to negate some of the disadvantages of an interpreter, but the argument for a compiler will always exist. To my knowledge there is little or no work being done in this area at the moment, and a preliminary inspection indicates that it will not be an easy task. In my opinion, however, the availability of a compiler for APL will be a major step in the maturing of the language, and will greatly increase its usefulness.

Another major problem is workspace size. No matter how large the partition is allowed to become there will always be the application that presents you with 'WORKSPACE FULL'. The ideal solution would be a variable workspace as implemented at the University of Alberta - and I'm sure that we'd all like to type)CLEAR 256k. Unfortunately, most institutions do not have, or do not plan to have large machines dedicated to time-sharing. In most cases APL will be running on machines that are primarily used for batch processing. An alternative solution that has been used with some success here at York and at the University of Laval is to create an overlay structure in the workspace. Functions and data are loaded and erased under function control as they are needed instead of being limited to only those items present in the active workspace. A third solution (which has also been considered at York) is the availability of a batch APL processor which would use the same library as the time-sharing system. In this case, a program, or workspace which has been tested and stored under the time-sharing system could be executed under OS as a batch job in a large region. This may seem to be a quick and dirty solution to the problem, but in my opinion the option of using the batch system offers the APL user greater flexibility than a strict time-sharing system.

One important development which two versions of APL (Sharpe and York) both offer is the ability to access data files on disk. I will not dwell on this as other speakers will be describing file I-O in APL. The important point is that no system can be used for serious computing if its I-O is closed unto itself. The ability to access standard OS files allows APL to communicate with other systems and programs written in other languages. The ideal APL system, and one that I would like to see in the future would carry all or these features; file I-O, a batch version and/or a variable workspace setting, and a compiler feature.

At this time two questions arise; given such a system what are its applications; and even without these developments, just how can APL fit into the educational computing environment? One of the main uses for APL has been as a teaching tool for mathematically oriented subjects. As a student-computing system the developments I have just discussed are not necessary. The major need to insure greater student use is cheaper terminals. Time-sharing is not the least expensive way to provide student computing but it does seem to be the most effective. APL is best suited to the needs of students who whish to use a computer as a problem solving tool yet have no desire to have an understanding of the mahcine itself. One of the main problems, (if you can call it that) with students accessing APL is that they tend to do much more computing than they would on a batch system, due to the interactive nature of APL. This characteristic makes APL suitable for the learn-by-independent-experiment approach. In schools and lower level university courses APL provides a simple yet potentially powerful computing facility. The use of packages will increase, especially in the areas relating to statistics. APL has not and probably will not be of interest to the teachers of Computing Science as the language makes the computer very invisible. I have however heard of a course given at the University of Alberta where first year computer science students are taught APL as a first language. The idea is to initially teach the concept of algorithms and to introduce the computer as a device for solving problems before placing restrictions and constraints on the students as they work back into machine language. This approach has been considered a success, and it is my opinion that a programmer trained in this manner would be more creative and efficient.

There is increasing interest in the application of APL for computer assisted instruction, CAI. APL allows greater freedom in the design of computerized lessons than most CAI author languages. Two other features of APL that make it an attractive medium for CAI are the ease with which one can modify and debug programs written elsewhere. With the great amount of time necessary to create CAI materials, the sharing of lessons between installations is very important. The wide spread usage of APL allows this. Dr. Miencke will be speaking this afternoon about his experiences in this field. The main limitations of APL for CAI are workspace size, and the fact that some character manipulations are slow in the interpreter. There is an interesting article in the January 1972 'Communications of the ACM' concerning a new CAI system, the NEWBASIC/CATALYST system. The software used is an improved version of BASIC which is modified to be an "algorithmic interactive interpreter" - in other words it functions in a manner similar to APL.

To the researcher, APL has provided a facility to write and execute programs quickly to debut program logic before coding an algorithm in FØRTRAN. There is increasing interest in the use of APL as a matrix manipulation language. If compiler facilities, or a batch system were available, I'm sure many more researchers would be coding their programs in APL. The main objection of scientific programmers to APL at present is again the slowness of the interpreter for some applications, along with workspace limitations. Because of the wide range of operators available in APL, the system is drawing the attention of researchers involved in simulation of systems. Dr. Medow will be talking on this application in his paper. With the implimentation of access to OS files, APL will become useful to the researchers as a system for collecting and massaging data from experiments. The time saved by programming applications in APL in many cases will outweigh any penalty caused by the interpreter.

One of the stated aims of the original implementors of APL was to produce a tool for systems design. They have succeeded, and the adoption of APL as a design tool for business systems will be one of the uses administration will find for APL. File I/O has greatly expanded the potential usefulness of APL in business. Mrs. Seaburg will be discussing applications to data processing in her presentation, so I will concern myself with only one potential application. Real-time data entry and information systems are expensive and time-consuming to produce. Pilot systems can be set-up in APL with little programming time to evaluate the proposed system. In some cases, (especially if a compiler exists) it might even prove advantageous to leave systems in APL rather than place several real-time systems on the institution's computer. APL can also be used to train data entry clerks prior to the real-time system being operational.

There are two final points I would like to make before closing. APL is symbolic and therefore not dependent on the English language. This fact is important in a country such as Canada where bilingualism is becoming of greater importance. APL has the potential to be a medium of exchange of algorithms and programs between institutions in the two main cultures in this country. The University of Ottawa, as a bilingual institution, has already experienced this acceptance.

Because programs written in APL tend to be short and modular, there are greater possibilities within the language to adapt functions to your own purposes written by others. Unfortunately there is currently no organization to get APL users together to exchange ideas and co-ordinate the distribution of functions. The Universities have a possible framework for the distribution of functions. The Universities have a possible framework for the distribution of programs (CLING and CSIC), but I think we could all benefit from the formation of an APL interest group. Perhaps through meetings and conferences such as today's, we can open the channels of communication to form such an organization, to investigate better uses for this language in which so much interest is being shown.

BUSINESS APPLICATIONS IN APL

BY

Charlotte Seaberg
I.P. Sharp Associates Ltd.
APL Consultant

For the past two years I have been a consultant to a major Canadian company based in Toronto. During this time I have had the opportunity to apply APL to a wide variety of problems ranging from statistical studies to the development of a large integrated planning model of the entire firm.

AS you know, APL has stirred much controversy within programming circles, and there has been some opposition to using APL for business applications. For this presentation, I felt it would be informative to trace the utilization of APL within this particular firm. While the approach taken to solve the particular problems encountered is specific, the problems are representative of those found in most, if not all, companies.

This company is in the business products industry. Figure (I) shows how four major functional areas of this firm are related. The Marketing function is responsible for direct sales, establishing sales targets, tracking sales performance, and setting marketing strategies. The Service function installs and maintains the business machines at the customer's site. The Distribution function sets inventory levels for each product. It also is responsible for tracking the flow of machines as they go from the warehouse to the customer, or when they come back from the customer to either the refurbishing center for repairs, or to the reconditioning center to be rebuilt or retired from the system.

The Finance function, in addition to developing annual plans and producing various performance analyses, prepares a monthly forecast. The forecasted data is prepared in conjunction with the other three functions and is directly related to their activities. The activities to be predicted are numerous. And each activity has to be forecasted for the next 12-month period. Examples of some typical projections are shown in Figure (II). The forecasts are critical to the corporation because they provide output which forms the basis for many key management decisions.

THE INITIAL SYSTEM

The initial APL project was done for Finance to assist in the forecasting task. Previously, the forecasting effort was performed manually. And, as a result, suffered from several shortcomings. First, considerable time and human resources were required. Approximately three man-weeks were allocated to this task each month, with only sufficient time to generate one complete forecast. Second, the forecasting logic was not specifically defined, with the result that the logic used from month-to-month was not always consistent.

The manual procedure was as follows: After Finance had prepared the forecast, the functional managers would convene for a review meeting. Changes to the forecast were made on the spot. If, for example, Marketing felt that orders for a particular product were too low, the adjusted order figure would be linked in. Often there was not time to investigate the impact such changes might have on other activities. The result was that, oftentimes, the final forecast was not internally consistent.

The Finance manager felt that a simulation model of the company should be developed which would have two important objectives. First, it would produce the required forecasts. Second, perhaps more importantly, it would provide management with a means to test various alternatives, try different strategies, and determine the results. Management would be able to answer such questions as "How would different pricing decisions affect profits?" or "How can increases in productivity be achieved?" Such a model would also help management view the business as a system. For instance, the marketing manager might increase orders to see the resulting change in total revenue. However, other effects, such as changes in commissions, increases in inventory levels, increased demands on service manpower, resulting from the order change would also be apparent.

The APL simulation model was developed in the following way: Historical data was carefully examined to determine the company's organization and operating responsibilities. Next, the important variables were identified. Then, the relationships between these variables were defined. Lastly, an attempt to quantify these relationships was made.

Approximately two man-months of effort was spent in developing and programming the model. Figure (II) shows the initial system. The simulation model, or equations of corporate logic, had been completed. Inputs to the model consisted of assumptions, parameters, and strategies which the manager could control. The simulation inputs consisted of about 30 variables which are similar to those listed in Figure (II). These inputs, including such variables as growth rates, trade rates, inventory levels, and various fixed and variable costs, together with the model equations produced about 300 output variables, such as net profit, product revenue, expenses, and manpower.

As an indication of the types of relationships that comprise the simulation model, Figure (III) depicts a hypothetical set of variables which describe one of the outputs, namely, the number of orders delivered in any one month. Figure (III) shows that orders delivered is directly related to available manpower, the current order level, the unfulfilled orders or order backlog, and to the availability of products in inventory. Orders delivered is also shown to be indirectly related to such variables as sales productivity, production capacity, shipping delays, and prior average sales.

After the simulation model had been in production for several months, the forecasting procedure radically changed. Now, instead of waiting for the entire forecast to be completed, before having the management review meeting, the managers would assemble in the terminal room during the initial simulations. Each manager would be paying particular attention to the inputs and outputs which directly affected his own area. The managers, together with the financial analysts, would decide jointly on the various inputs in order to produce a forecast which was optimal for each of their individual requirements.

This particular model was very economical. Each forecast required about 8 to 10 cpu seconds. After management had become fully conversant with the model logic, an average of four to six simulations were run to produce each 12-month forecast. The time involved to produce a final forecast had decreased from three man-weeks in the manual system to an elapse time of two days using the APL simulation model.

This was the first introduction of timesharing capability within the Canadian company. Many of the managers were excited about the changes the APL model had created and there was much interest in expanding the use of APL to other areas.

SYSTEM AT ONE YEAR

Figure IV shows the APL system, representing about eight man-months of effort, one year later. By this time all of the other major offices of this corporation, in the United States, were using APL, and specifically, the simulation model, on a production basis.

In Canada the simulation model was still the core of the system but now represented only 60% of the total APL usage. Another major development was a financial model written in APL by one of the financial analysts. The output from this model included the traditional financial statements, profit/loss statements, balance sheets, and cash flows, which were available for any projected period up to seven years forward and for any historical period.

Finance also had the important responsibility of preparing a two-year plan. The current planning effort utilized a large batch planning system, running on a Univac 1108, which had been implemented by the American company. The 1108 planning system was run in a simulation mode. That is, dozens of test plans, or simulations, were run over a three-to-six month period before the final plan was derived. Each simulation required extensive input data which was derived manually. To simplify this process APL was used to assist in developing the required input data. The APL functions proved to be tremendous time savers and also established a foundation for the APL planning model which was to be developed the following year.

During the first year, the data base increased ten-fold to about 1000 K characters. Previously, the data base contained only region data. Now it had been expanded to contain selective data at the branch level as well. An elaborate reporting/performance system was developed to allow selective and/or exception reporting for various comparisons between actual, budgeted, and projected data at both the region and branch levels.

In this company, like many other companies, the data required by one area considerably overlaps that required by other areas. For instance, Service may require Marketing and Distribution data, while Finance may utilize Accounting and Personnel data. So, as the data base became more comprehensive, many areas found that the data they needed, on both a routine and a special project basis, was, more often than not, available on the data base. This fact was the impetus for other areas beginning to use APL for their own special studies. Not all of the resulting applications involved large models. Some were exceedingly simple, involving elementary matrix operators and the use of a report generator.

Figure V shows how some of the functional users were applying APL to their own problems.

Marketing found APL useful for deriving productivity analyses, generating sales reports, and for evaluating various marketing strategies. For example, very recently all of the Canadian sales teams were divided into two groups. One group received special training which was felt to have a positive effect on sales, while the extra training was withheld from the other group. APL was used to track the performance of the two groups over a six-month period, to evaluate the results by various statistical tests, and to present the final results in report and graph form.

Service has used APL for manpower planning and budgeting; to assist in location of tech reps; and to analyze the status of machines which are in need of new parts for either efficiency or safety reasons.

Detailed depreciation schedules, showing the accumulated depreciation for all machines, are manually prepared by Accounting. Various depreciation techniques exist; some of which are difficult, if not impossible, to implement without the aid of a computer. APL was used to produce both projected and historical depreciation schedules using any desired method of depreciation.

Since Distribution is concerned with tracking the machines as they flow from the various distribution centres to the customer and back again, APL has been profitably used to determine optimal shipping routes, and to prepare freight cost analyses.

THE SYSTEM AT EIGHTEEN MONTHS

Figure VI shows the APL system at eighteen months. The three major developments to the system were a long-range planning model, a generalized forecasting system to be used primarily as an adjunct to the simulation model, and conversion programs to transfer large masses of data from the company's in-house computer to the APL timesharing system.

The initial APL routines used in conjunction with the 1108 planning system were refined, expanded, and incorporated into a large-scale, on-line planning model. This model is similar, in many respects, to the simulation model. It affords the user the same degree of control over the output and can be utilized to test various alternate courses of action. The planning model has proved to be a successful approach and has become an integral part of the planning effort throughout the corporation.

It should be stressed that the simulation model cannot be construed to be a forecasting model. It merely simulates the flow of business activity; in essence showing the expected outcome if certain parameters are assumed. The model inputs, as with any simulation model, greatly influence the reliability of the output. Many of these inputs were tediously developed each month by several means; visual inspection, simple averaging, examination of historical trends, etc. These methods were both time-consuming for the analyst and were also highly error-prone.

As a result, a generalized forecasting system was developed. This system, using the short-term forecasting technique of adaptive forecasting, includes several smoothing models used in conjunction with various optimizing techniques. Presently, about 70% of the critical variables, such as order levels, productivity, and revenue expectations, are being forecasted by this method, while the remaining variables are under the control of the manager. The effect of the forecasting system has been to reduce the forecasting error 20 to 40% and to allow forecasts to be derived in seconds rather than hours.

The data base, in the Canadian company, had doubled to about 2000 K characters. Much of this data was produced by the simulation, financial, planning, and forecasting systems. However, about 25% of the data was being input via the 2741 terminal. During this period, cobol programs were written to allow 360/40 tapes, containing highly summarized general ledger, billing, and sales analysis data, to be read directly into the on-line APL files.

THE EXISTING SYSTEM

The existing system is shown in Figure VII. The two central features are still the simulation and planning models which currently comprise about 40% of the Canadian APL usage. APL is increasingly being used by businessmen, in "program" or "calculator" mode, to produce various reports and analyses.

Enhancements to existing systems are continuously being made. For example, monte carlo techniques are being applied to the Financial model to provide output based on probability distributions rather than on single-point estimates.

To date, all of the APL models have used logic based upon data, concepts, and relationships internal to the company. Because the external economic variables plan an important role in business, and economic model, to interface the existing planning models, is under development. Sub-models will include a system to retrieve historical economic data for analysis, projections of key economic variables, and various statistical and econometric routines.

SUMMARY

Both the simulation and planning models have received considerable attention because they do represent state-of-the-art applications. The success of this system, I feel, can be largely attributed to two factors. First, it is due to the Finance management of the Canadian company who initiated the use of APL and greatly influenced the direction it took. And secondly, it can be attributed to the APL language. APL allowed the existing system to be developed in 12 to 15 man-months through the efforts of a consultant and a financial analyst. The use of APL spread rapidly throughout the company, primarily because businessmen found APL allowed them to effectively utilize a time-sharing facility without becoming programmers. Since business is dynamic, any business application system must be flexible enough to handle the countless additions and revisions that occur frequently. APL greatly minimizes the problems in this area.

Figure VIII shows a list of selected time-sharing applications which are in current use. Those denoted by an asterisk are being utilized by this company, using APL. All of these and similar ones not mentioned, can be classified as "management or operational aids". Although they differ in complexity and sophistication, all of them provide assistance to the manager in the area of corporate "decision-making".

Many forecasts of time-sharing usage in this decade indicate an increasing trend for these types of applications. I feel that APL has established itself as a "desirable", not just a "feasible", language for business and, in the near future, other companies will discover the potential APL offers for on-line decision making.

APL IN BUSINESS

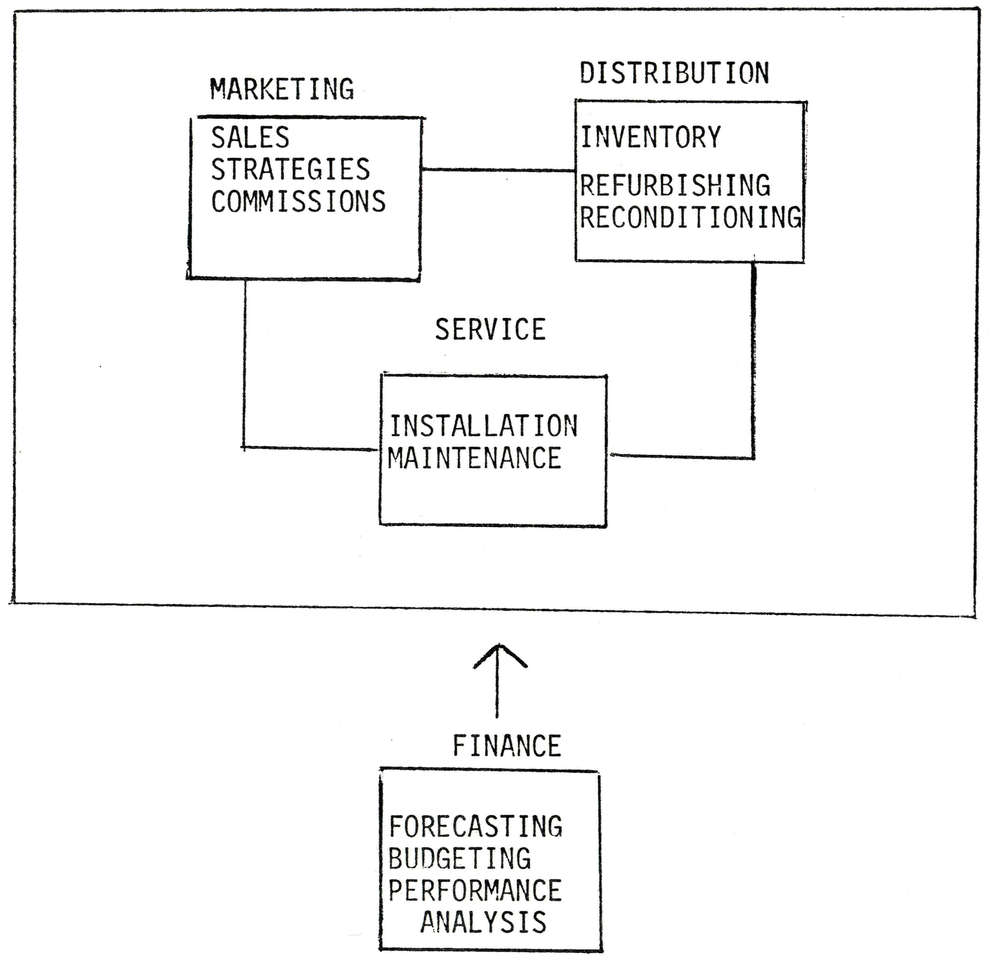
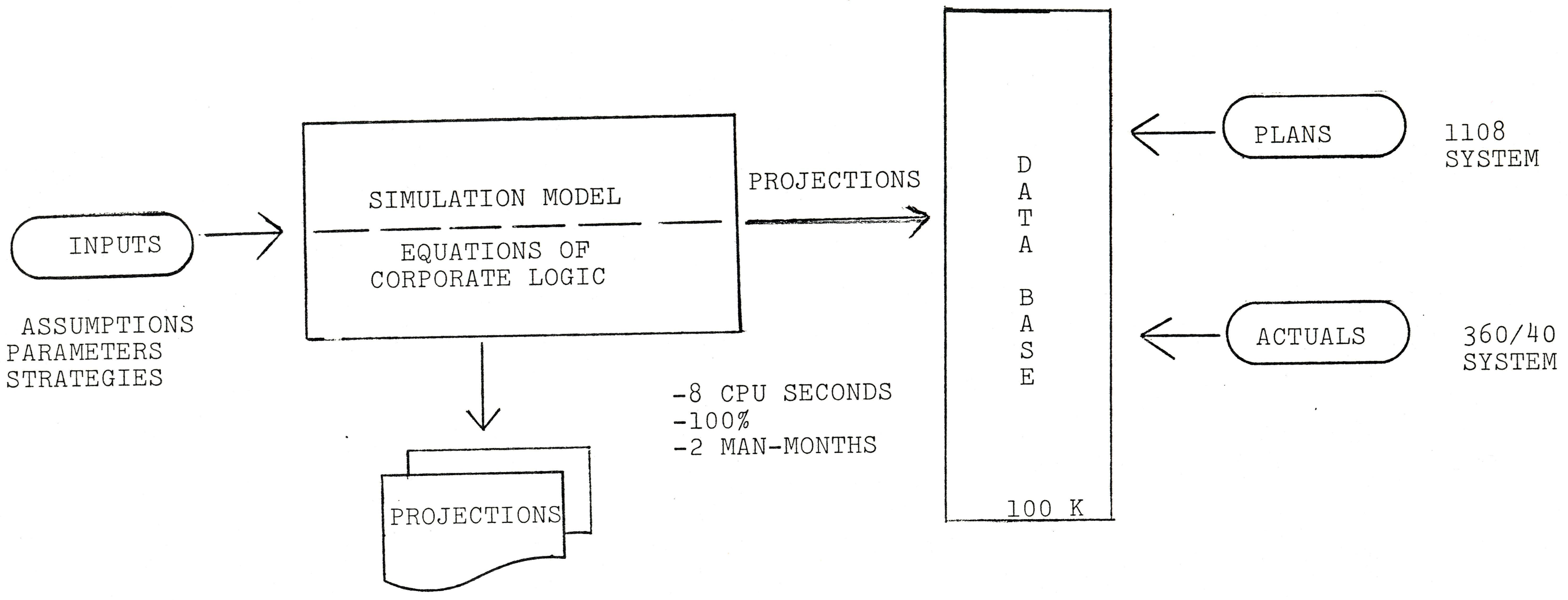


FIGURE 1

APL IN BUSINESS

INITIAL SYSTEM



SIMULATION INPUTS (N30)

GROWTH RATES
TRADE RATES
INVENTORIES
ADVERTISING
FIXED COSTS

SIMULATION OUTPUT (N300)

NEW ORDERS
TRADE ORDERS
REVENUE
NET PROFIT
COMMISSIONS
PARTS EXPENSE
MANPOWER
ORDERS PLACED

FIGURE II

APL IN BUSINESS

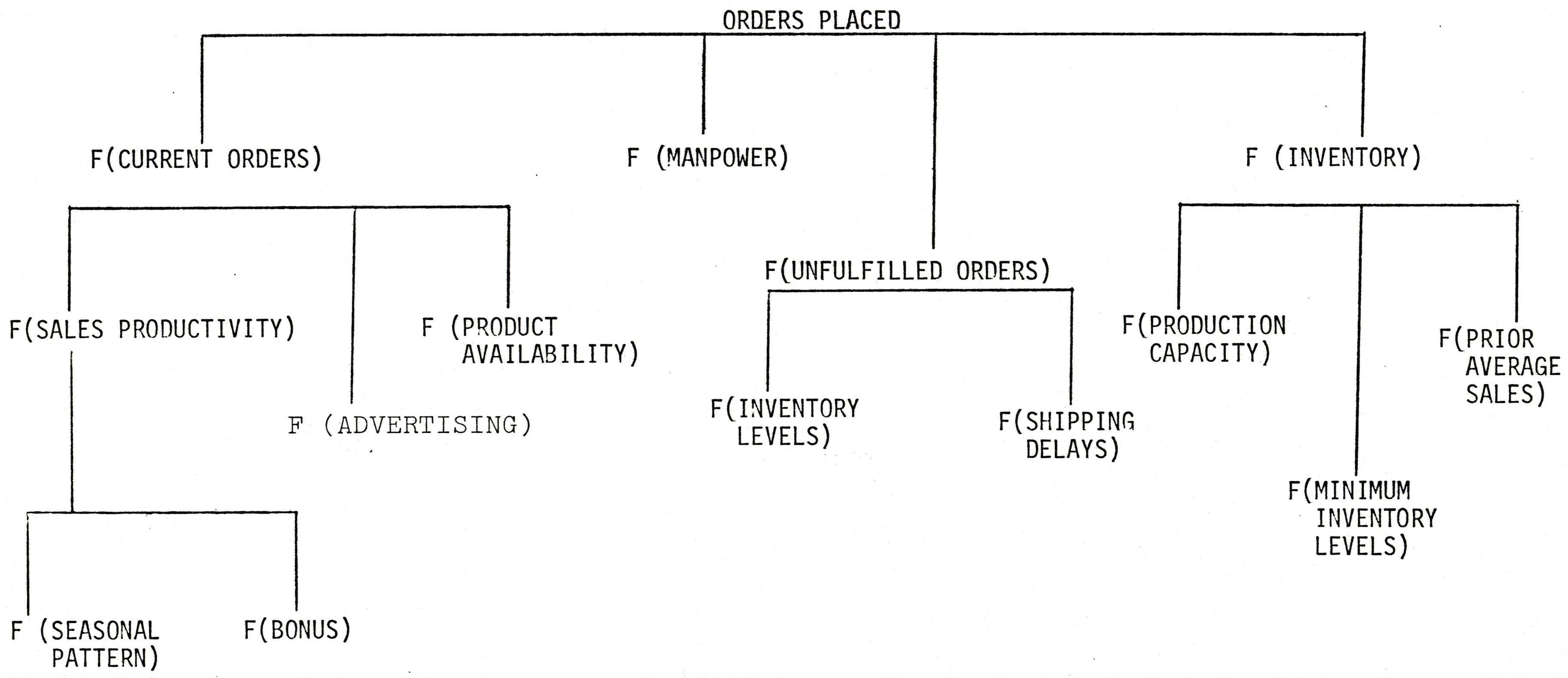


FIGURE III

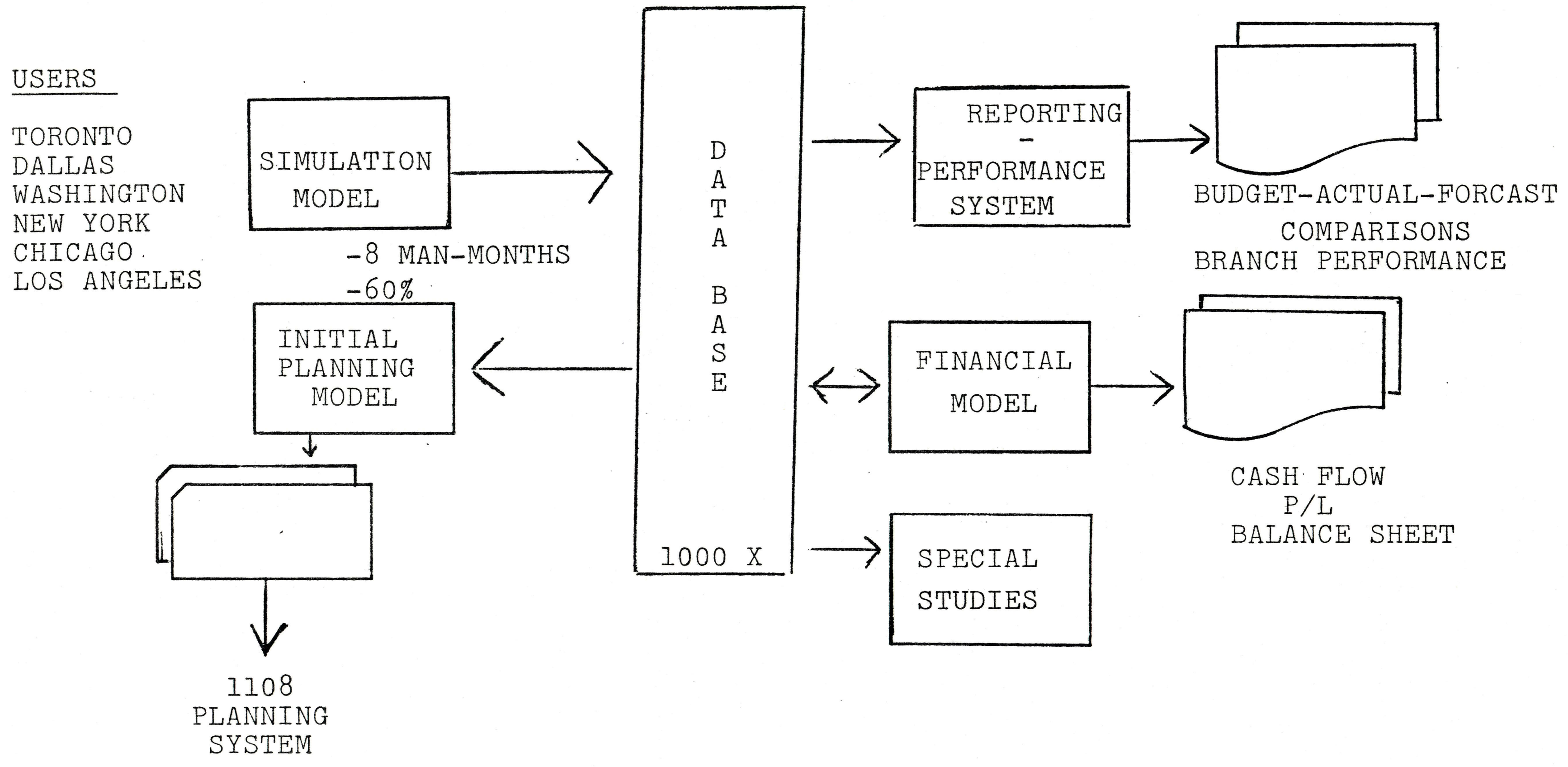

USERS
TORONTO
DALLAS
WASHINGTON
NEW YORK
CHICAGO
LOS ANGELES
SIMULATION MODEL
-8 MAN-MONTHS
-60%
INITIAL PLANNING MODEL
1108 PLANNING SYSTEM
DATA BASE
1000 X
REPORTING - PERFORMANCE SYSTEM
BUDGET-ACTUAL-FORCAST COMPARISONS
BRANCH PERFORMANCE
FINANCIAL MODEL
CASH FLOW
P/L
BALANCE SHEET
SPECIAL STUDIES

FIGURE IV

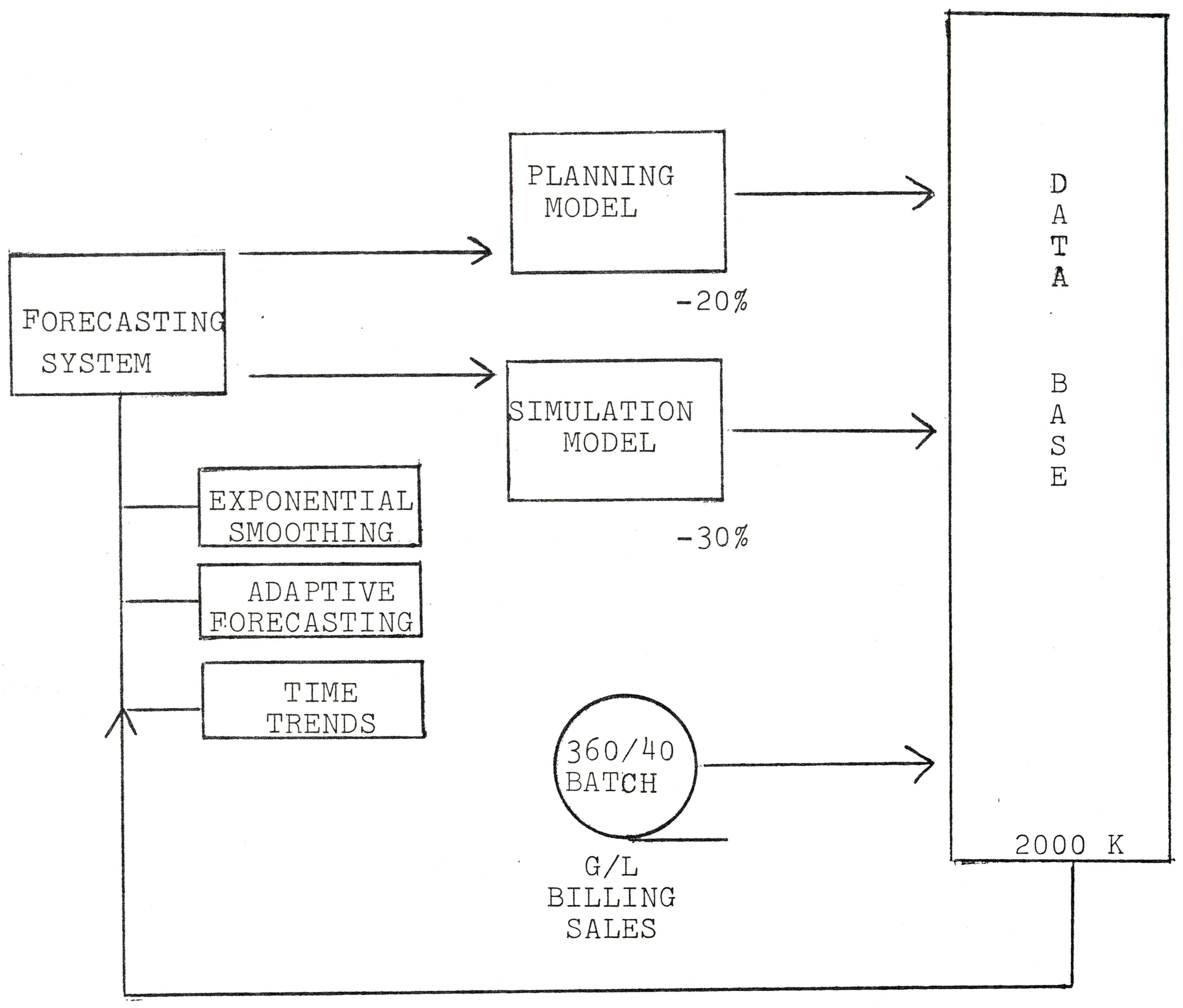
APL IN BUSINESS

MARKETING	-PRODUCTIVITY ANALYSIS -SALES ANALYSIS -MARKETING STRATEGIES
SERVICE	-PARTS ANALYSIS -MANPOWER PLANNING -TECH REP LOCATION
ACCOUNTING	-DEPRECIATION ANALYSIS
FINANCE	-FINANCIAL RATIO ANALYSIS -SPECIAL - PROJECT ANALYSIS -SALES TEAM PLANNING
DISTRIBUTION	-FREIGHT COST ANALYSIS

FIGURE V

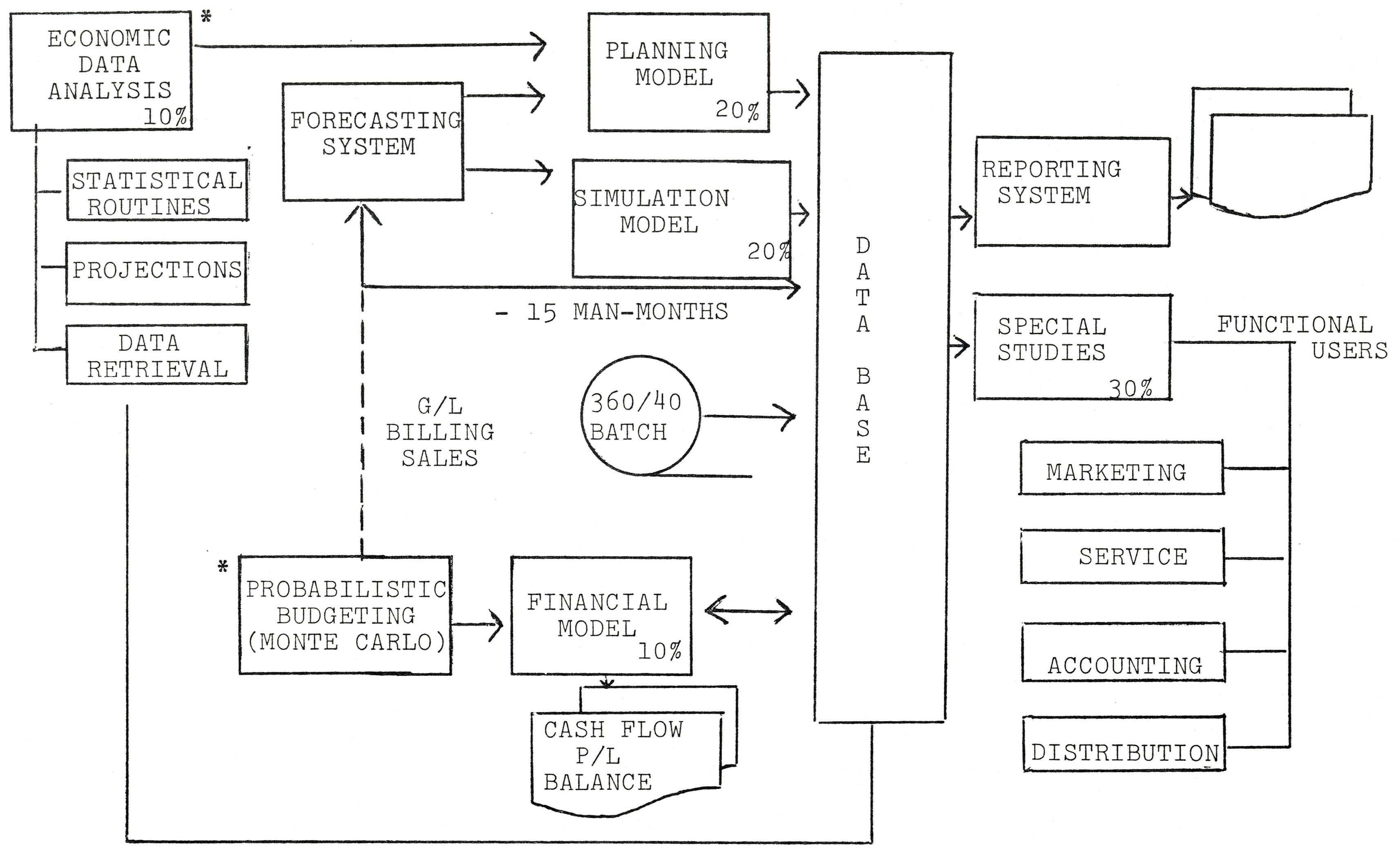
APL IN BUSINESS

SYSTEM AT 18 MONTHS



APL IN BUSINESS

EXISTING SYSTEM (24 MONTHS)



* UNDER DEVELOPMENT

FIGURE VII

APL IN BUSINESS

ACQUISITION - MERGER ANALYSIS

BUDGETING *

CORPORATE MODELLING *

DEPRECIATION ANALYSIS *

ECONOMIC FORECASTING *

FINANCIAL STATEMENTS *

FORECASTING *

INVESTMENT ANALYSIS

LEASE-BUY ANALYSIS

MARKETING STUDIES *

PERSONNEL SCHEDULING *

PORTFOLIO EVALUATION - SELECTION

PRODUCT PRICING ANALYSIS

PRODUCT PROFITABILITY *

PROJECT EVALUATION (DECISION-TREE ANALYSIS)

RISK ANALYSIS

SALES ANALYSIS *

VARIOUS TECHNICAL ANALYSIS PROGRAMS *

* UTILIZED BY THIS COMPANY IN APL

FIGURE VIII

THE INTERNAL STRUCTURE OF A CENTRALLY EXECUTED INTER-TERMINAL COMMUNICATION OPERATOR

BY

P.I. Medow
Dept. of Economics
York University

In considering the APL instructions that are relevant to a centrally programmed process of interaction among remote terminals it is useful to refer to a particular experiment. But first the corresponding acts of inter-terminal communication must be clearly seen as elements of program execution.

a. The role of a master communications terminal in monitoring inter-terminal communication elements of program execution

An algorithm may be represented as an operator graph with branchings. Its operators may include any transformation process that combines and transforms input signals in a specified way.

The great diversity of such operators, which exceeds by far those specific transformations that conform to algebraic rules, presents a major challenge to representatives of substantive disciplines. In particular, it is possible to employ operators whose switching circuits represent contingency tables, extrapolation methods, direct simulation, and pattern manipulation techniques, including the manipulation of text data.

A common element in such circuits whose formal properties have received insufficient attention concerns the hierarchical structures to which they may be reduced. The basic operations associated with the restructuring of trees and with their filtering are employed explicitly in the heuristic operators associated with artificial intelligence and with the study of linguistic forms, as well as in the storage and retrieval of data. But there exist many other possibilities for their application, particularly in structural studies of technological systems, of information systems, of organizations, of banking institutions, and of social institutions generally.

Beyond the use of operator circuits that are internal to the computer the possibility exists of incorporating a wide variety of external physical "black boxes" into algorithmic operator graphs. These can be not only other computers, mechanical and electric devices, and data measuring or monitoring installations, but also organisms including human organisms, and their responses may then be carriers of operational energy signals (e.g. directed to other machines) as well as of information signals (e.g. directed to a computer).

In the specific case in which such an external automation or black box is a human operator coupled to a remote terminal the following types of responses may be distinguished: (1) those that are produced by human operators acting as pre-programmed transformations (responding according to earlier instructions); (2) those produced by simulated role players (responding in the light of their own acquired knowledge, judgment, and intuition, but in relation to fictitious or at least non-operational situations); (3) those that are produced by "on-line" role players (engaged in real situations).

Beyond this, in those cases in which role players are of the learning type, their responses may be preceded by secondary acts of consultation and discussion, as well as of interaction with artificial or real learning environments.

The corresponding remote terminals may then be employed for such auxiliary purposes.

Two types of program instructions are needed to execute mixed-mode algorithms of this type: the familiar ones that create internal switching circuits; and similarly structured external circuits of interterminal communication.

Since both types of circuits are produced by a single algorithm, it is appropriate to assign both functions to a single service terminal. In relation to its external function such a terminal then operates as a master communications terminal, in the sense that all substantive or operator terminals can access each other only through the central communication algorithm and hence only in accordance with the constrants that it specifies.

b. A substantive application: the river cities game

An important aspect of the adaptation (or "embedding") of technological systems, institutions, and processes to wider social norms concerns the selection of sites for production facilities, particularly when these are known sub units or "branch plants" of larger functionally integrated complexes.

In particular, if several international corporations each design and adopt for long-term implementation one integrated complex of ten to twenty distinct production units, there still remains a significant combinatorial problem in deciding how many individual branch plants of various types are to be located in particular candidate countries. Yet from the point of view of an individual country the cumulative impact of four or five such decisions over fifteen or twenty years may define the difference between a plantation or mining economy and an economy heavily engaged in the production of electronic equipment, or, from a different point of view, between an economy whose capacity to meet vital foreign exchange requirements will be relatively stable and one that will subsequently become permanently dependent on foreign loans.

Similarly, from the point of view of individual territorial units within a nation (e.g. urban or regional settlements), a comparable cumulative impact may result from location decisions of public program management agencies.

It follows that in such a context any agency that is able to make explicit and display to lower level authorities the relevant auxiliary dimensions of available combinatorial alternative will in fact provide a significant institutional basis for a partial adaptive process.

The communications requirements of such an adaptive institution may be defined formally in terms of a homeostatic game in which territorial automata reject centrally proposed (initially indifferent) decisions of higher level. But they may also be initially defined in terms of a relatively simple "operational reference image" such as the following one: Let there be a river on whose banks there are three cities.

Let the industrial area of each city (and also closely associated agricultural production centres) be located on the river's left bank, while its residential area (together with its social infra-structure, and government) is located on the right bank. Let the basic task that confronts each local (city) government concern the rejection or acceptance of centrally defined trial proposals for locating particular combinations or "packages" of additional branch plants on their left banks in the light of a) a local long-term plan of social development for the right bank and b) of basic "external" commitments.*

The limited numbers of structural distinctions conveyed by this image are sufficient to permit a replication within a centrally monitored multi-terminal laboratory of a computer-assisted adaptive institution of this particular kind.

c. The technical basis of interterminal communications and the external execution "unquote" operator

Normally, the transmission of a signal from a remote terminal to the central operating unit is effected by pressing the "return" key. Subsequently the response of the central unit is conveyed through an appropriate activization of the movements of the typing ball. The transmission of a signal to another remote terminal, however, is effected in a different way, namely via an external storage disk employed as a communications medium. Thus an act of transmission requires two coordinated events: an act of storing information on external disk by the broadcasting terminal, and an act of retrieving from disk by the receiving terminal.

---

* A local criterion that is then likely to acquire considerable importance concerns the extent to which such proposed "location packages" would facilitate the attainment of a "socially balanced" composition of professions within the local population. A city of miners would presumably tend to reject proposals for additional mines.

More specifically the broadcasting terminal must execute the operating system instruction)SAVE, and the receiving terminal must execute the system instruction)LOAD immediately afterwards. Similarly, the return response to the broadcasting terminal requires that each of these instructions be executed a second time, beginning, this time, at the second terminal. Accordingly, the program providing for the execution of a single "external" operator activates simultaneously marginal sub-programs at two terminals that incorporate acts of control over peripheral equipment (the disk) that are coordinated in time. This last requirement is met by repeating periodically the receiving terminal's attempts to retrieve material from the disk space of the broadcasting terminal until attempt is successful. More specifically, in order to avoid error messages (which would interrupt the entire program) the receiving terminal retrieves a binary indicator (a switch) at each trial retrieval cycle until a change in that indicator informs it that the material that it seeks is in fact available.

The capacity of each parallel sub-program to automatically control peripheral equipment is provided by the external execution or "unquote" operator(1).

(a) The external execution ("unquote") operator:

The "unquote" operator is an instruction that temporarily suspends the execution of a program and then inserts a new line of instruction before returning to execution mode. In particular, the line of instruction that is inserted can be an instruction referring to the peripheral equipment of the operating system, such as)SAVE or)LOAD. More generally, however, such a line can be constructed from both literal and numeric characters that are initially catenated into a single literal ("passive") expression in such a way that they correspond to an executable line of instruction. The non-literal ("active") counterpart of such an expression is then executed separately before the normal execution of the program is resumed(2).

---

(1) Initially developed at York University for the York Terminal System. (YORK APL)

(2) Since local variables are erased whenever execution is interrupted, the variables whose names appear among the literal characters to be "unquoted" should not be specified as local variables.

The general properties of the unquote operator are illustrated in Figure 1.

```
      P←4  5  3  2

      Q←5

      OP←'↑ρ ι↑⌈'

      I←3

      ⍎'R←';P;' ';OP[I];' ';Q

      R
  4   5   3   2   5
```

The 2 last line produces in effect

```
        R ← 4 5 3 2 5
```

When the external execution operator is employed to store on disk a program or a variable whose name has been defined as NEW in the course of program execution, the act of storage is thus effected by the instruction:

```
        ⍎ ')SAVE'; NEW
```

The corresponding act of retrieval by a receiving terminal is similarly effected by the instruction:

```
        ⍎ ')LOAD'; NEW
```

In order to insert delays of specified duration between attempted retrievals of information, from disk, however, such instructions must be followed by a line that also combines the "idle" operator with the "non-carriage return" operator.

(b) <u>The "idle" operator and the "non-carriage return operator"</u>

The "idle" operator delays by approximately 1/15 seconds the execution of the next instruction. It does this by

✓

causing the remote terminal typing ball to be actively engaged in responding, but without printing. This disconnects the terminal from the central operating unit for that time.

It is written:

```
Q ← '80'⌶ 30
```

To extend the duration of the idling period a vector of such operators is defined. A delay of one second is thus produced by the vector:

```
Q ← 15⍴ '80'⌶30
```

But since the execution of such a vector of fifteen idle operators will also move the typing ball fifteen spaces to the right, in cases in which the length of such a vector exceeds 130 characters (the width of a page) this will normally automatically produce a carriage return. This must be neutralized by a "non-carriage return operator", which is written:

```
N ← 'AO'⌶30
```

Accordingly, an idle instruction designed to last approximately 20 seconds will be written:

```
Q ← 260⍴ST ← (129⍴ '80' ⌶ 30), ('AO' ⌶ 30)
```

d. A model block-oriented language with external operators

The possibility thus exists of constructing a single non-primitive operator that will first establish a continuing broadcasting relationship between the central communications terminal and specified operator terminals for each external sub-block of a large model, and then activate individual pairs of acts of broadcasting in accordance with the specifications of a central algorithm.

*NOTE: In York APL ⌶30 converts a left argument which is a character string containing hexadecimal representation of a string of characters to the characters.

COMPUTER ASSISTED LEARNING IN APL

BY

Dr. Meincke
Erindale College

Erindale College opened its doors in 1967 with somewhat of a spirit of innovation and a desire to explore as many of the avenues opened by developments in educational technology as possible. Computer Assisted Instruction was one of these avenues but resources were not available to run systems such as the IBM COURSE WRITER. The U. of T. computer centre at that time was implementing an inter-active time-sharing system called C.P.S. which was based on PL/I. A preliminary study of the CPS by IBM for us at Erindale indicated it might be suitable for CAI, but nothing much happened until Mr. George Horner of I. P. Sharpe Associates came along to Erindale to demonstrate a system called APL-CAT, which he had developed based on a similar system in use at Orange Coast Community College, called CAL-APL. It seemed very clear to me at that time that this was exactly what we needed. However my knowledge and experience with CAI was minimal, my knowledge of computers much less, so I was basing my judgement on several rather general factors: minimum cost, system reliability, ease of development and maximum utilization of equipment. Now these are very broad aims and they don't mean very much, but these are the kinds of things that everybody dreams about when they try to build a new system. So it was decided at that time to use APL-CAT on the I.P. Sharpe System.

I wrote a few programs and these were used by a number of students on the one 2741 terminal that Erindale had acquired, but I was tremendously impressed by the simplicity of the system and was in fact writing programs within one hour. (now everybody claims that for APL---I still don't know any APL but I was using APL-CAT in less than an hour. It takes you an hour to learn how to sign on the terminal but after you've done that, you are fine.) But for some reason I was never able to persuade many of my colleagues to write programs. I kept dragging them down to the terminal to show them what I had written, and they were all duly impressed but then went away, never to appear again.

Then two magnificant things happened. First of all, David Vaskevitch appeared on the scene, and second the new director of the U. of T. Computer centre, Dr. John Wilson, decided to implement APL at U. of T. In the last year, David has written about 25 CAI lessons, and has developed the systems functions to the point where I believe that they are second to none in North America. (We have even had two people over from Heidleberg to look at the system). In addition, while he was a student working on this during the summer, David also performed numerous other programming tasks. In fact almost all of the work to be described today is due to David. (Incidentally this is his first year as a student formally registered at the University of Toronto).

APL is now up and running at the U. of T. with a fantastic degree of reliability and all the lessons that work are stored in a public library and may be accessed by the students on all campuses. There are now 3 2741 terminals at Erindale with students lining up to use them. I think it is safe to say that Computer Assisted Learning, using APL is going strong at the U. of T. Certainly nothing has been found to cause us any regret at all for having chosen APL. In fact, every new project reveals another facet of APL which is incredibly well suited to this function. My amazement at the success of APL is only equalled by my amazement at the reticence of my colleagues to explore the possibilities of this new technology.

Now, I would like to tell you about a few of the details of the present system and outline some of the lessons so you can see how APL has been used. I'd like to compare APL and Course Writer for a simple drill program. For the very simple things, APL-CAT takes 1 more statement than Course Writer, but APL-CAT is immensely better for more complicated lessons.

Here is a simple drill written in Course Writer III.

COURSE WRITER III

```
qu What is 2x3?
ca 6
ty Right
wa 5
ty You added.  Try again
un Wrong   Try again
```

The first statement defines a question that is typed out to the student: "What is 2x3?" The next line indicates the correct answer is "6"; type "Right" and go on to the next question. The line "wa 5" indicates a wrong answer, and the comment "You added, Try again" is typed to the student. Automatically the student is given another try. If you want to branch somewhere else, a branch instruction must be placed there? For an unanticipated answer (un) the comment "Wrong! Try Again: is typed and automatically the student is branched back to try again. If you wish the student to do something else, you have to put in a sub-operation code. The following will show just what this would look like in APL-CAT.

<u>*APL*</u>

```
Q1:GETN 'WHAT IS 2×3?'
→R1 IF MATCH 6
→W1 IF MATCH 5
'WRONG ! TRY AGAIN'
→Q1
W1:'YOU ADDED! TRY AGAIN'
→Q1
R1:'RIGHT'
```

You can see it is a little bit longer, and looks a bit more complex, but what you can do in APL is write a conversational program to interact with the instructor to build this sort of lesson. Here is how it actually works. Question one is indicated by the label "Q1" "GETN" simply prints out "What is 2x3?" and reads the numerical value input by the student. There are two types of read allowed, numerical and literal input. The next line says branch to R1 if there is an exact match to 5. The beautiful thing here is that you have the full capability of APL. The student can type in 6, 6.0, 6.0E0 or any numeric input. In fact, if he is smart he will find out he can type in 2x3 and get the right answer. If both matches are unsuccessful you branch the student back to "Q1". This is a catch-all phrase which covers all unanticipated answers. "W1" is obviously the comment for an anticipated wrong answer and the student is branched back to "Q1". The line labelled "R1" is the comment for a right answer, and you can continue. That is roughly the kind of sequence one might use in APL.

However, you can write even more complicated lessons. "MULTIPLY" is a program I wrote for my young daughter who is needed some help in multiplication.

```
∇ MULTIPLY
 [1] R1:A←?12
 [2] B←?12
 [3] 'WHAT IS ';A;' TIMES ';B;' ?'
 [4] READN
 [5] MATCH A×B
 [6] →R2 IF TRUE
 [7] 'SORRY ';B1;' TRY AGAIN'
 [8] READN
 [9] MATCH A×B
 [10] →R2 IF TRUE
 [11] 'THAT''S STILL NOT RIGHT ';B1;' THE ANSWER IS ',A×B;'TRY THIS'
 [12] →R1
 [13] R2:'THAT''S RIGHT, NOW'
 [14] C←?12
 [15] 'WHAT TIMES ';B;' IS EQUAL TO ';B×C
 [16] READN
 [17] MATCH C
 [18] →R3 IF TRUE
 [19] 'SORRY ';B1;' TRY AGAIN'
 [20] READN
 [21] MATCH C
 [22] →R3 IF TRUE
 [23] 'THATS STILL NOT RIGHT. THE ANSWER IS ';C;'SEE IF YOU CAN DO THIS'
 [24] →A1
 [25] R3:'THATS RIGHT NOW'
 [26] →R1
 [27] A1:A←A[A⍳?100]
 [28] 'WHAT IS ';A[1];'-';A[2];' ?'
 [29] READN
 [30] →A1 IF MATCH -/A
 [31] 'WRONG'
 [32] →A1
∇
```

It is simply a random number exercise. Generate a random number with the APL function "?" and put it in A., another random number and put it in B. The question asked is "What is A times B" with the random values substituted. "READN" gets the answer, and then "MATCH AxB". Checks to see if it is correct. Again this answer processing capability; you don't need specific numbers. You can do all your calculations right in APL and do the match. This function AxB could be any complicated mathematical expression that APL will take. In addition you can write subroutines and you will see some more complicated ones in a little while. You can see by following through that the student is given 2 chances, then if the second attempt is wrong the comments "That's still not right. The answer is ----- (the value of AxB)" and you just put out the answer of AxB. The flexibility again is enormous. Then it goes on to drill in division and so loops back and forth. If you wanted to do a random exercise in CAN-4, the language created by O.I.S.E. the following describes what would be involved.

<u>CAN4</u>

```
2002 RV,V39
     CO,V40,(200×V39)/2
     T1,V40
2001 RV,V39
     CO,V41,(200×V39)/2
     T1,V41
     IF,V41,GE,V40,2001
     CO,V42,V41-V40
     T  WHAT IS @V41 - @V40
     U
     CN
     IF @A,EQ,42;↑2001
     T ,WRONG,
     GO,2002
```

The first 8 statements essentially generate two random numbers. Then you have the question "What is" the contents of buffer 41 subtracted from whatever is stored in buffer 40 and so on. The equivalent exercise in APL is much shorter. You can see that as soon as you start writing anything in the way of a "page-turning" exercise you gain a fantastic amount in just the size of the program.

Here are some of the operations codes available in Course Writer III:

COURSE WRITER III
SOME OPERATIONAL CODES

MAJOR	MINOR	OP CODES
qu	ty	ld
rd	au	ma
pr	ep	cc
ea	fp	if
cm	continue	em
aa	ed	br
ab	pa	
ca	fn	
cb	ad	
wa	sb	
wb	mp	
un	dv	
ny	ld	

There are major operations codes which can double as statement labels. The operations can include questions, reads, answers and many other things. The minor codes occur within the major op codes and indicate messages, actions etc. There are op codes for other operations, like loading numbers into buffers and various other operations. The point is that in one summer, a similar yet easier to use set of operations codes have been developed by one man in APL. This is a list of most of the operations codes we use in APL-CAT:

SOME OPERATIONAL CODES

START
STOP 'LESSON NAME'
'TEXT'
READ
READN
MATCH 'X'
SCAN 'X'
OR 'X'
AND 'X'
BEFORE
BLANK
IGNORE
REPLACES

START is the first statement in the lesson and initializes counters and a sequence to get the student's name and eventually look up records. That's all the professor has to know. "STOP" appears at the end of a lesson. Text is delineated by quotation marks. No special symbols, characters or codes are needed. Of course quotation marks can get you into trouble. If you forget to close them, you end up putting in a large amount of text which is really a part of your program. "READ" reads in literal data which "READN" reads in numeric data, again with the complete APL capability. "MATCH 'X'" requires an exact match. "SCAN 'X'" looks for the occurrence of a set of letters "X" in s character string. That's done fairly easily by setting up a matrix of the input from the student and checking to see if any column of that matrix contains what you are looking for. SCAN can be misleading but it is also a very powerful operator for negating the effects of spelling errors in the student's reply. "OR" and "AND" are used in answer processing capabilities.

In the Example:

→ L1, L2, L3 IF SCAN 'A; or 'B' or 'C' BEFORE 'D' or 'E'

L1, L2, L3 are line labels. The student is branched to

L1 first time through, then L2, then L3. There can be any number of labels if for example you would like to pass on progressively greater hints. "IF SCAN" implies we look to see if in the answer sequence the series of characters 'A' or 'B' or 'C' occurs before 'D' or 'E'. For example if you are looking for "FOOT POUNDS" as the answer to a question such as "what are the units of work" you can look for "FT" or "FOT" or indeed you can ignore the "o's" and just look for "FT". The remarkable thing is the amount of answer processing that can be carried on in one line of coding. Again with our example of units of work, using the "BEFORE" function you can scan for the occurrence of "FOOT" or "FT" before "POUNDS" or "LBS" or "PDS". A very powerful set of answer processing capabilities.

The following is a list of the programs we have created to date:

PROGRAMMES NOW IN USE

APLCATHOW1
APLCATTWO

PROGRAMES NOW IN USE (cont'd)

APLTHREE
STUDENTHOW
LIMITS
CALCULUS1
CALCULUS2
CALCULUS3
MOTION
CIRCULAR
CIRCULAR2
MICHELSON
MORLEY
WORK
WORKA
LOADLINE
LOADLINETRANS
EANDM
UNITS
RELATIVEXPT
WFFNPROOF
SHAKEAWFF
GAME1
GAME2

"STUDENTHOW" teaches the student how to use the terminal, access programs in the library and take a lesson. "APLCATHOW", "APLCAT2" and "APL3", (not very imaginative, I guess) are three programs that actually teach the programmer how to write lessons in the APLCAT language. Then we go on to actual teaching programs. We are concentrating here on the first year student, to aid in bridging the gap between the high school and university. We try to give a lot of background material for students who are having a rough time in first year. "LIMIT" discusses the concept of limits in the mathematical sense. I believe there is another program being written on this subject. The three programs on calculus teach differentiation and integration, concentrating on the practical aspects with as little theory as possible. We have a program on basic motion, circular motion, (CIRCULAR and CIRCULAR2) and a very interesting pair, MICHELSON and MORLEY. These concern introductory ideas in relativity, the Michelson-Morley experiment. These ideas are always introduced in first year physics, but these programs give the students a chance to play with this idea for better understanding. Then we have two introductory programs in work. One of the questions that always concerns students is the satellite going around in perfectly circular orbit doing no work. This is explained to a certain extent in this program. When I was teaching an electronics course I wrote LOADLINE. This is a design problem for field-effect transistors amplifiers and LOADLINETRANS is one for one with a transformer coupling. EANDM is introductory electricity and magnetism and UNITS is drilling in UNITS (metres, kilograms, joules, etc.). Now this is a fun one - RELATIVEXPT, one I have just written for the first year lab. The student sits down at the terminal and believes he is at the controls of a linear accelerator. He can choose a particle which he may accelerate through a given voltage. It's absolutely trivial. The thing is not at all sophisticated but the students enjoy it very much. I liked the term used by an earlier speaker - "massaging the data" - that's just perfect. In relativity, if you get particles up to very high energies the mass increases and keeps on increasing as the speed approaches the speed of light but can never exceed the speed of light. You can show students a graph of this or send them home with a slide rule to calculate the mass at some speeds but students have difficulty grasping these concepts. Sitting down at this lesson, the student can really get a feeling for these equations and the laws of relativity. The student can see when the mass becomes important. With calculations to nine significant digits, he can see that accelerating an electron through a potential difference as low as one volt introduces a change in the mass out in the ninth digit or so. Whether this is accurate or not is another question but the demonstration is effective. This last group of programs represents a real "tour de force" in programming by David. (WFFNPROOF, SHAKEAWFF, GAME 1 and GAME 2). What this does is teach the student the introductory ideas of propositional calculus (or symbolic logic). It's done in what is called a "Polish Notation" and deals with well-formed formulae. We have a demonstration of this coming up.

Let us just look at RELATIVEXPT again to show a little bit of the programming involved here because this is essentially all there is to this lesson.

```
∇ RELATIVEXPT
 [35] SETV:'V ?'
 [36] READN
 [37] V←B0
 [38] KE←Q×V
 [39] M←(KE÷C*2)+MV
 [40] S←C×(1-(MO÷M)*2)*0.5
 [41] V;'         ';KE;'      ';M;'       ';S
 [42] →SETV
 [43] STOP 'RELATIVEXPT'
∇
```

Lines 1 to 34 which are not reproduced here are essentially literal output data which tell the student what to do. He also chooses a particle up there too. The essence of the whole thing is from line 35 down. The student is queried for a voltage which is input through READN. Whatever is in the read buffer is assigned to V. Kinetic energy (KE) is calculated as QxV (Q was assigned in the earlier part of the program). The mass is calculated (M) and so is the speed. These are easy things to do in APL but very difficult in any of these other languages. Line 41 simply prints out the values and the student is branched back to SETV to set another voltage. That is the heart of the whole program which can keep them going for an hour or more. They just type END when they want to.

(There followed a demonstration of WFFNPROOF and SHAKEAWFF on the 2741).

The main point about these programs (WFFNPROOF and SHAKEAWFF) is that they are generating a completely random set of characters and the program can analyze this completely random set of characters. What we have is a new kind of situation where the student can sit and ask a question such as whether or not something is a WFF (well-formed formula). You can see this in the example in figure 10. The computer is essentially thinking along with the student - something possible with APL but not the other systems.

Lessons now in preparation are the following:

LESSONS IN PREPARATION

Lorentz Transformation
Time Dilation (Bob Hillis)
Statistics (Bill Kurmey)
Student directed Physics Problems
Logarithms
Limits
Lessons in Cobol
Math Drill

Some of these are waiting only for files to be implemented on the U. of T. system. Some of these are being written by students as projects. In the student directed physics problems, I'm trying to set up a physical situation, list all the parameters in that situation and let the student choose which ones he wants to know. This allows the student to drill in those processes he wishes. The MATH DRILL is being written by David in such a way that the student will be evaluated all the way along as to the number of correct answers and wrong answers and he will be branched to particular levels of difficulty depending on his performance.

I would like to list some other possibilties:

SOME OTHER POSSIBILITIES

Genetics Experiments
Language Drill
Chemistry - Reaction Kinetics
Electronics Circuit Analysis
Simulation Games

I feel these are rather interesting. With genetics experiments you could set up a whole fly population and get immediate population distributions as the situation develops. In chemical kinetics you can simulate fast or slow reaction. These are just a few of the possibilities that come to mind.

Projects under development (not lessons as the previous items were):

PROJECTS UNDER DEVELOPMENT

TRANSLATORS
PROPOSITIONAL CALCULUS
CANONICAL PARSERS
PROCTOR (YORK APL AT RYERSON)
FILES
CATALOGUING
KWIC INDEX
ADMINISTRATIVE SYSTEM
STUDENT RESPONSE ANALYSIS

We have translators under way which will be able to translate lessons from the CAN-4 programs into APL-CAT to take advantage of some of the programs written in this language. Translators are also being written for other languages like Coursewriter. We are planning much more interesting things in propositional calculus. The Canonical Parser would be a very great advance in the state of the art. It would be able to take a formula answer, parse it into pieces and analyse the formula put in by the students. These things can be done now in simple ways but in fact this thing should make it much more flexible. One system David has written is called PROCTOR using York APL at Ryerson. This uses the unquote operator developed at York to let the student have the computer query him and load the lesson he chose. The PROCTOR program acts like a proctor - it helps the student find his way through the library of programs.

Files are coming up at the U. of T., and once we have them we will be able to store student responses and improve the programs. Cataloguing is another essential feature of this whole thing. David has built up a conversational cataloguing system so that once a lesson is written the author is asked to catalogue it in a conversational way. He is asked for the title, where the lesson is stored, and then to write a short paragraph which is available for Keyword in Context (KWIC) indexing. This is the next program David has written. Keyword in Context index manipulating programs allow the student to find out what various programs deal with. Other things that we are looking at right now that are of great interest are possible use of APL in administrative systems and also student response analysis programs for the files.

To summerize then, these are the things that I think are great about APL. First of all the mathematical capabilities. The development flexibility is really the major feature. You can write specialized functions which you can store in the workspace and tell the student about these when he loads that workspace. You don't have a precise and rigid system on top of the actual lesson programs. The workspace concept is a unique and different way of storing and presenting CAI materials. The equipment is basic equipment and is used for a large number of other purposes. The 2741 terminal is expensive but, as was pointed out, more and better things are coming along. In fact there is the possibility coming for using teletype and teletype compatible terminals. These devices do not support an APL character set, but we don't need them for the CAI applications. The equipment can be expanded and a lot of work is being done on this at Orange Coast Community College. The 2741 has been hooked up to a graphic display (an X-Y Plotter). The other interesting thing here is that they have hooked up the terminal to a random access micro-fiche viewer. This contains about 48,000 slides of material. The terminal (program) can randomly access any one of 48,000 pages in less than 4 seconds. Here again there is a fantastic capability. Just think of what 48,000 pages represents!

Sophisticated answer processing is another thing which is part of the system designed here. The other important feature is that it's transferrable. Everything is stored as literals. When you get a listing it is all literal data. Once I heard about some work going on at McGill, I asked for some details of the programming and what was returned to me was an assembler language listing. Something which was of no use to me. The idea of learner control is very important, as is generalized operations. You can generalize the programs to a fantastic extent as for example in the WFFNPROOF programs where you have something that is interacting with the student in a very real way. I would suggest that this is getting on to a sort of artificial intelligence, but I don't like to use that phrase because of its ramifications. The thing is you do not have specific answers in mind. You really are operating on the answers that are given or questions if you like to phrase it that way. David likes to say an answer is a question in this particular context. That is one of the very powerful features of this system that has been developed.

Well, that is the summary and as I said at the beginning, we have absolutely no regrets for having chosen APL, and in fact, as far as we can see, there is no real limit to the horizons that we are able to see at this point.

"The Use of A.P.L. in Teaching Mathematics at the University of Waterloo."

BY

Dr. Roden
Cambrian College

I want to talk for a while about some of the experiences that we had at the University of Waterloo. As Mr. Simpkin mentioned, I've been up at Sudbury since last month, but it's my Waterloo experience that I'll be talking about and I'll also throw in a few of my own biases and prejudices as I go along. The topic here is "Use of A.P.L. for teaching Mathematics" and I should perhaps point out that what I'm talking about is completely different from the sort of application mentioned by the previous speaker (Dr. Meinke). We were not using A.P.L. for C.A.I. at all. In fact you might say when I'm finished that we didn't really teach with A.P.L. at all.

Let me explain to you what we were doing. We are talking about teaching mathematics students, and in my mind there is really not much of a division between programming and mathematics. I would contend that every mathematics student should take some programming just because of the logical organization and discipline involved in writing computer programs. Given that we want to teach programming to our mathematics students, how do we go about it? We felt that A.P.L. was one very good means of doing this for a number of reasons. First of all it's a very easy system to learn. Whether the student has had any programming experience before or not, it's generally a matter of less than an hour before he can start writing A.P.L. programs. The characteristics of the A.P.L. language are partly responsible for this. There are no obstacles such as dimensioning and formatting, or integer and/or real values and so on. The student can get right down to work. We found a few little problems, however, which messed up some of the students. One goes something like this: we tell the student he can use the terminal as a desk calculator, so he types in something like

```
2 + 3 x 5
```

and he gets back a result: 17

Fair enough. So we tell him also that he can have numeric expressions and alphabetic expressions and that alphabetics are designated by quotes. So he types in something like this:

```
'ABC GEORGE AB CHARLSE
```

and he forgets the last quote. He presses the carriage return and nothing happens. The student might scratch his head for a minute and then he tries typing in something again:

```
2 + 2
```

and nothing happens. He might try

```
2 + 3 x 5
```

which worked before but still nothing happens. So he tries clearing his workspace - if he knows how to do that. It doesn't work, of course, so he kicks the terminal or something like that! He can try to sign off or send a message to the operator, but nothing works, as the system considers it is receiving text. This is one of the problems students have which should be fixed. Maybe when the system is receiving text it should put out a message after 5 or 6 lines just to say, "I'm listening to text. If you want to get out, type a quote."

This problem was probably the most common one. The poor kid could very well be having a nervous breakdown by this time. He thinks the machine is picking on him! There are other kinds of funny things that can happen. For example you can get into an input loop in a program, where the program asks for input, maybe executes a little, and then goes back to the step requesting input. To the guy at the terminal it seems that all the programme ever does is ask for input. And it does no good to hit the attention key while the programme is waiting for input. You have to intercept the programme while it is executing. It doesn't matter how fast he goes; the student hits return and he can't hit attention before the computation is done. Other than silly little problems like these, we found that the students progressed very quickly.

A.P.L. is very nice for learning how to solve problems because it removes so many of the mechanical details from a lot of things. Say a student wants to do matrix operations. It is so nice to be able to solve a set of equations and not have to do it with three sets of nested DO loops as you need in FØRTRAN. This allows the student to get on with learning the concepts. That's my opinion, anyway.

There are other characteristics of A.P.L. which I feel are suited to teaching mathematics. The logical structure of the language is something that I feel is very important. Ken Iverson's idea that the whole structure of A.P.L. must be ultra-logical has been followed faithfully in the implementation. This is evident from reading A.P.L. manuals or any of the correspondence which goes on between the implementors and users of the system who would like to see it improved. When decisions are made to add things to the language, often the basis used is "is this a logically consistent addition?" Also the elegance and economy of the language, the fact that you can do so much with so little, is important. To make a program efficient you want to write it as elegantly as possible. The fewer lines in a program, the faster it is going to execute, generally. A.P.L. teaches students to think about expressing things concisely and logically, something which languages like FØRTRAN certainly don't.

There is also the time-sharing aspect of the system. This is very important. Because it is a time-sharing system and not a batch system, the students can get pretty close to "hands-on" control of the machine. It would be nice if we could let them right in the computer room, but at Waterloo we had between two and three thousand Mathematics students so this was not at all possible. With the terminal at least you feel as if you have control of the machine, even though the systems designer knows you really don't. It is responding to what you ask for. There appears to be a direct correlation between what you type in and what you get out. This kind of thing really encourages the students to experiment and I think this, idea of exploring things and following ideas through to their logical conclusions, is the whole essence of learning things like mathematics and science. As Ken Iverson likes to say, when a student is working at a terminal with A.P.L., if there is something he doesn't know, the solution is to try it.

We decided at Waterloo to use A.P.L. as a tool to aid the students in the study of Mathematics and, as I alluded a few minutes ago, the title of this paper is perhaps a misnomer. We made A.P.L. available as a tool and let the students explore things on their own. Rather than set up a lot of things for the students to do on the terminals, we made it fairly optional. We rented a /360 (at that time the only thing A.P.L. would run on) model 40. The University already had a /360 model 75, but at that time the only version of A.P.L. ran under DOS and the model 75 used OS. Also, the Mathematics Faculty wanted to have its own machine, so that the Computer Science Department could have something they could play with to do software research on without messing up the model 75. The idea initially was that during the 40 prime shift hours per week, A.P.L. would be the number one thing going on the machine. Outside the prime shift hours, other people could have the machine if they needed it and if not, then we let A.P.L. run anyway. What actually happened was that we ran 40 "guaranteed" hours a week. Of the remaining 120 hours a week, generally A.P.L. was running for one hundred of them. However, there was no operator around and, if the system ran, that was fine. If it went down, too bad. This was how we ran.

To make the system available to the undergraduates, we set up a terminal room, and for a start we put in twelve 2741's. Without giving the students any formal instruction whatever, we just opened up the room and let them use it. Any student who was interested enough to come and ask for an account number could get one. We gave them a little manual that Paul Wilson and I had produced, entitled "A Taste of A.P.L.", which stepped the students through most of the elementary aspects of A.P.L. just to get them started. We found most students made out very well on their own, with the help of other people in the room. As was pointed out by the previous speaker (Dr. Meinke), when you have a few terminals together, people start getting buddy-buddy and offering all kinds of assistance to one another.

To illustrate the kind of response we got from the students, we plotted the number of terminal hours logged each month. Fig. (1). In the beginning, we had only 12 terminals and as we had started in May there was low activity during the summer. It started to pick up during the winter and peaked about March, 1970. There was an awful trough when we changed from the model 40 to a model 50 in September 1970, and also switched from DOS to OS. We all came pretty close to having heart attacks in the process! We were one of the very few people with OS A.P.L. at that time (number 2 or 3 in Canada at least). We tried it for a week or so and it crashed so many times we told I.B.M. we didn't want to see it again. (The bugs were promptly fixed and we have run OS-APL since October 1970). At the same time we added another 10 terminals to the undergraduate terminal room, so there were 22 terminals available during the 1970-71 academic year.

Originally the connections were dial-up, but these caused all sorts of problems. The expense of going from the terminal out to the city exchange and back into the computer room added many dollars to our bill. Also we were using Bell demodulators (data sets) in the computer room and these things, in addition to being outrageously expensive, had one weird little property. If the person at the terminal end did not sign off properly, the computer somehow never got wind of the fact that he was gone. So his port was not freed up. This brings us back to our previous problem. Whenever a student got fed up and hung up the phone, another port was disabled. Pretty soon we found that with only a few terminals on, everyone else was getting a busy signal. We hardwired the terminals and solved that problem, saving a lot of money in the process. Also it's a lot easier for the students to sign on now. All they have to do is turn on the power switch and they are connected.

Our peak month was in March of last year when we recorded 8,000 connect hours for A.P.L. Of those, 5,000 were used by undergraduates. It must seem that a room with about 20 terminals is an adequate facility, but during the winter last year those terminals were busy all the time. I am not sure what the experience has been lately as I have not been at Waterloo, but you can get an idea of the sort of thing that is going on by projecting the usage figures.

There were a few courses where students were given assignments to run in A.P.L., but for the most part they used the system as a matter of interest. In many cases, the terminals were used for calculations associated with assignments, but students weren't told they had to use A.P.L. A lot of them simply found this tool very easy to use.

Of course a lot of this time was spent fooling around. Students like to play games! Some wrote programs to play tic-tac-toe, and deal hands of bridge and all the rest. Actually some of these were pretty respectable programs. I don't think that this sort of exercise is out of line for a student as a part of his undergraduate training in Mathematics. For example, one student wrote a very elegant program to play 3 - dimensional tic-tac-toe against the person sitting at the terminal.

The students discovered all sorts of things. We didn't go out of our way to provide a program library specifically for the students but an extensive list of library programs was left in the terminal room and the students found out about most of them. They discovered a lot of things that we weren't particularly happy about as well. I might just mention one little thing that really caused us to chew our fingernails. I can talk about this now because it's been fixed, otherwise I wouldn't! It seems there was a bug, once upon a time, in A.P.L., where you could do something like this:

```
M←256 256 256 256⍴1
```

This simply says you want a 4 dimensional tensor, 256 rows in each direction, and each element will be represented by a 1 bit. Of course the A.P.L. interpreter shouldn't do any such thing. It turned out that the way the interpreter handled this was to check first to see whether the array would fit in the available core. To do that it asks itself how many locations does the array take. To find that out, it multiplies the dimensions together. So it does this and comes up with the result, which, when expressed as a double length binary number, turns out to look like:

```
100000000000000000000000000000000   (binary)
```

and the interpreter looked at only the low-order fullword. After all, who is going to create an array with 2^{32} elements in it? Well, our students will. The interpreter let them get away with creating this array and once they had done that, the whole world unfolded before their eyes. Once you have created an array, if you want to display some particular element of the array, all the interpreter does is look at the subscripts and ask whether they are within the range of the array and of course they are. So the user can address any bit in core that he wants, and he is not confined to his own workspace. Very quickly the students discovered where the status bits were which said whether or not a terminal was privileged. After they had figured out how to privilege themselves it was a short step to go on to bouncing other terminals. Then they discovered how to go all the way and execute a shut-down command. It was really disconcerting to sit at the operator's terminal and just watch the system fold in front of our eyes, especially after the guy had just sent a smart-ass message to the operator. This is fixed now however, and we have a nice solid system where, if the hardware runs, we can go for hours without a bit of trouble.

We supplied a computer and a bunch of terminals but as far as software support is concerned, we really did very little. As I said, we had an extensive library of programs on the system, but we didn't use them very much directly in the sense that we forced the students to use or look at them. We left them for the students to use if they wished. There were some things in the libraries that had some pedagogical benefits, I suppose. For example, Professor Pat Fisher wrote a system called CLASSPAK which was useful for keeping track of marks and things. Because of the power of A.P.L. his little package let you do all kinds of neat things like analyzing marks, looking at means and variances, cross-correlating from one question to another to see which questions are giving you a significant spread, and doing discrimination studies on the testing procedures - all sorts of things that professors would like to know about. As far as C.A.I. was concerned, we really didn't touch on it at all. There were some reasons for this. Mainly there just weren't any faculty around who were sufficiently interested to put in all the time and effort that is necessary to put a bunch of lessons together. There certainly wasn't much interest in the Computer Science Department itself. There were also some technical obstacles. Until very recently we didn't have any files on the A.P.L. system, so lesson size was limited by the workspace size. Also I think a C.A.I. course is of limited value if you don't have the facility to record the results as you go along. The best you can do without files is store a workspace so a student can load it, work in it and then save it again, but as soon as you give him the facility to save the workspace he's using, you've given him the power to do all kinds of nasty things. We'd had enough bad experiences in that direction. One thing I might point out is that there is some activity going on now in the area of preparing lessons in C.A.I. within A.P.L. We do have several packages in the library such as C.A.L. - A.P.L. and one that is based on the C.A.N. system from O.I.S.E. These are essentially sets of functions which made it easier to construct lessons. The work that is going on now uses raw A.P.L. programming rather than using any of these crutches. One thing that has intrigued me is that the lead on this has not come from the Computer Science Department at all but rather from other areas of mathematics. The one individual that I was in contact with in this area was one of the priests over at St. Jerome's College. This really shattered all my illusions about the conservatism of the Roman Catholic Church.

Well, I know you are probably thinking that this is all great, but what does it cost? I'd like to give you a few figures derived from our experience. I don't pretend that these are typical of anywhere else, but they are typical of our experience at Waterloo. First of all, there are many ways to figure costs in the computer game. You can juggle numbers around to prove just about anything you want. Let's look at a couple of extremes.

Let us consider an upper limit of the cost of providing this A.P.L. service. The yearly rental on the system (model 50) was $350,000 per year. Considering that we logged some 38,000 student hours at the terminals, this comes out to about $10 per terminal hour, which really doesn't seem too far out of line. It's certainly competitive with commercial rates. However there are a few things that colour this number. All I have looked at is the undergraduate usage. I have neglected things like faculty research, graduate students and the A.P.L. time we sold to sister institutions. So, let's look at another number. (I suppose you could call the previous one the ceiling and this the floor if you want to talk A.P.L.). Let's look at the two peak months, February and March. During those months the rental is about $30,000 but let's not assign it all to our undergraduates this time. What with other users on A.P.L. and background jobs in the system, plus the time when A.P.L. was shut down for Computer Science research, I don't think it's unrealistic to charge half the cost to undergraduate A.P.L. use. In fact maybe one half is high, but it seems like a good ball-park figure. If we consider that during those peak months we logged 5,000 undergraduate terminal hours per month, that comes out to a more reasonable figure, about $3.00 per hour. This covers operational expenses such as computer terminals and I think it also includes salaries. There were only a couple of us and we certainly were not well paid.

So this isn't expensive at all. The range is roughly $3.00 to $10.00 per connect hour. Now in a twelve week term we found that if a student is to get anything appreciable out of A.P.L. he should get something like 2 hours a week on a terminal. So let's say he gets 25 terminal hours in a twelve week term. Then we are talking of something in the neighbourhood of $75 to $250 per student for computing. Actually a reasonable figure is somewhere in the middle. The alternative would be to have them write FØRTRAN programs, and hope they will learn the same things by that route.

For a student running FØRTRAN programs to get anything out of his term's work, he ought to run at least 100 jobs through the machine. Of course the cost of running a FØRTRAN job depends an awful lot on the environment. About as cheap as you can find would be the WATFIV service on the model 75 where the cost per job is something on the scale of 20¢. So we are talking of $20.00 a term for FØRTRAN experience as opposed to something approaching $100.00 a term for A.P.L.

The question that immediately comes next is "is A.P.L. worth it?" Personally I think it is. The educational value of a good time-sharing system that encourages exploration, learning, and so on is so much greater that, to me, it is money well spent. Of course you must have the money in the first place, and that is a consideration. Actually the answer at Waterloo was to go halfway. Although we decided to make A.P.L. available to undergraduates, to keep the thing manageable we made a rule that only students in their third year and above could get their own account numbers. This kept the usage down somewhat. The first and second year students could get onto the system by some general numbers, but they couldn't save any of their work, so that somewhat degraded the experience for them. The real limitation was not terminals or C.P.U. but disk space. If we had given every student his own number and allowed him to keep a few workspaces, we would have needed another 2314 disk unit.

I'd like to list a few of the areas where I feel a system like A.P.L. is particularly useful.

First of all, there is Numerical Analysis. I think it's really ideal to be able to experiment with things, particularly if you are dealing with processes where you're taking an approximation to an answer and then refining your approximation in successive steps. The students can try out different methods and see which ones converge more rapidly, and which are the ones that take forever and can be discarded. In other words they get a feeling for what it means to go after a few extra significant digits of accuracy. These things can be very easily demonstrated in the classroom with a terminal. Rather than just explain what the methods are you can watch them work during the course of the class.

Simulations encompass so many things. Because A.P.L. is a time-sharing system you can watch the simulation as it proceeds and if initially you did not provide the parameters you wished, fine - stop the thing, make a few changes and let it go again. In a batch system if you make a mistake you end up with pages and pages of useless garbage. It's really nice to be in control and be able to experiment with combinations of parameters. There have been a few simulation packages produced at Waterloo and in particular Paul Wilson produced one as part of his Master's thesis which is along the line of G.P.S.S. for simulating queuing systems. You can get into some pretty complicated simulations with this. Paul used it to simulate the queuing that goes on in the operating system.

Teaching logic circuits is another good application. Eric Manning used A.P.L. for this. A.P.L., he found, was quite easy to use, as the boolean operators were a part of the A.P.L. language. There were no packages to write. Of course things like games theory, strategy for games and eventually operations research are all good applications. You can simulate different strategies and see how they work. You can play a person against the computer or by getting two terminals going you can even play the machine against itself if you're clever.

Statistics is probably one of the prime areas where A.P.L. can be useful. Our Statistics Department at Waterloo was probably our heaviest user. Most of the courses that did assign students problems in A.P.L. were in statistics under professors such as Jack Robinson and Cliff Young. I think there is a great deal a student can learn by using A.P.L. to study statistics. My experience has been that most people who have taken courses in statistics, even at the end, really don't know what statistics is all about. You learn a lot of formulae but just don't grasp the essence of it. All one has to do is look through the scientific journals for a paper where the results of an experiment have been put to a statistical test. My experience has been that 9 times out of 10 the experimenter really didn't understand what the test meant!

TERMINAL CONNECT HOURS

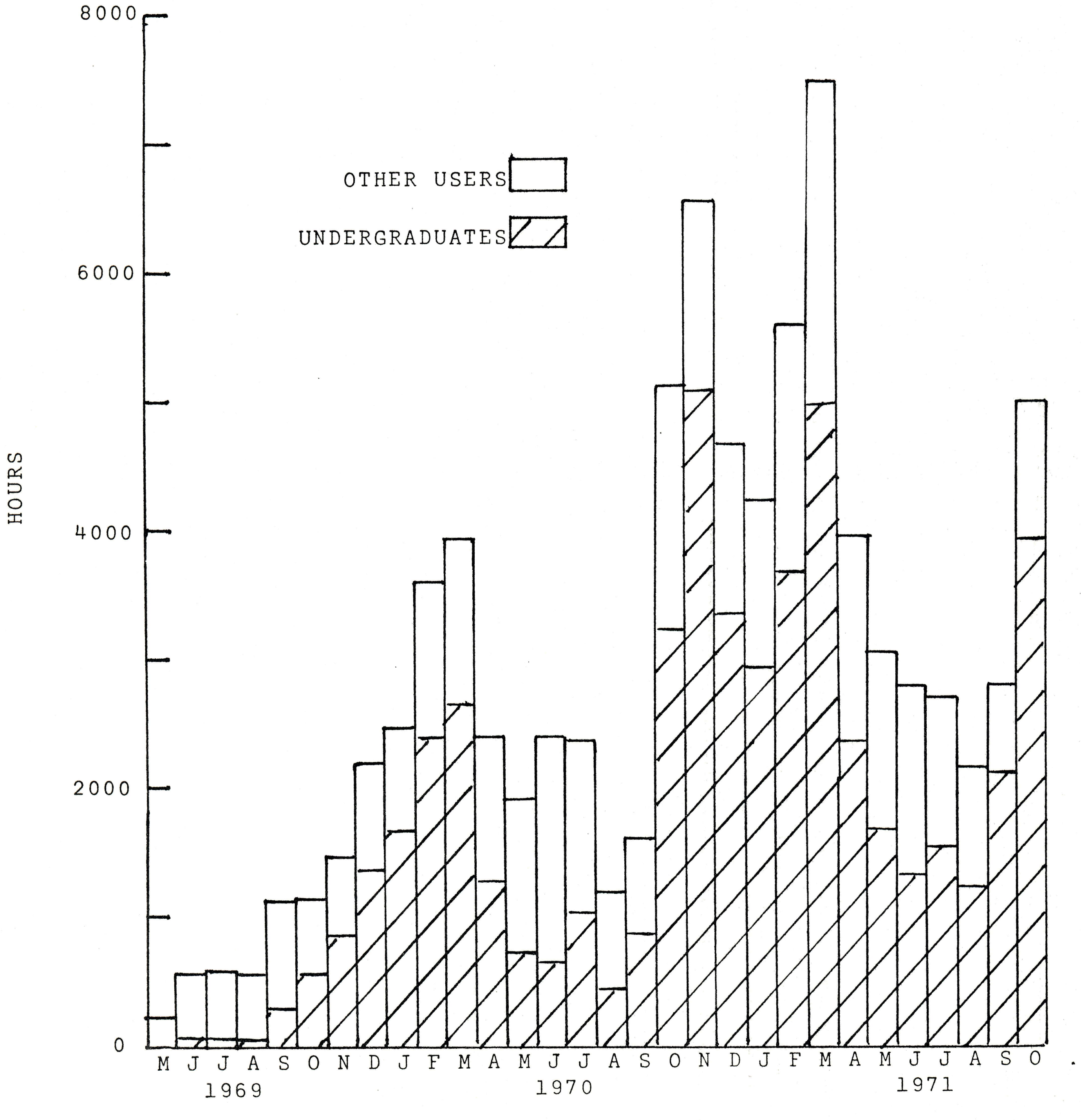


FIGURE 1

FILE I/O IN YORK APL

BY

Bob George
York University

INTRODUCTION

APL has been confined to an active workspace and restrictive library space and structure for long enough. APL is a programming language designed for use as a teaching tool that can communicate concisely and clearly an algorithmic solution to the computing needs of teachers and students. However if the computerized implementation of the language is to be a means of communication among people then it must allow ease of access to data base information created in the real world; for it is this information that people wish to analyse and draw conclusions from. Many researchers in mathematics, science, engineering and the humanities are presently finding that APL is a concise time saving language in which to define their algorithmic solutions. One who uses computer programmes usually has large volumes of information to be manipulated.

If one assumes that a problem is to be solved via computer analysis many questions must be answered. Three of these questions are:

A) Are there any constraints, within the language selected for algorithmic definition, that have a derogatory effect on the efficiency of the final algorithm?

B) Is the terminal hardware associated with the language suitable for the application?

C) Does the language provide access to existing data based information?

Let us look at APL in this light. To answer the question regarding efficiency, there is obviously a problem. Because APL is a procedure oriented language implemented as an interpreter it allows a person to debug and test a program more efficiently than any other computerized language. This is a significant advantage for APL because most of the human resources are spent in the debugging and testing of programmes. Therefore APL allows effective administration of human resources. However remember that I said APL was implemented as an interpretive language. This automatically implies that APL does not in all cases produce efficient machine code corresponding to the APL function definition. In some cases it produces the optimal coding; such as for matrix products, matrix invertion and the solution of linear equations. However in cases where optimal coding is not implied by the language, inefficiencies may creep in. These inefficiencies become increasingly significant when one begins to manipulate larger and larger volumes of data that are made available in data bases. Hence the problem of efficiency depends heavily on the organization and size of any data base accessed via APL.

APL is extremely dependent on restrictive hardware. The hardware dependency is clear. The machine normally used for development work is both slow and noisy and hence not conducive to a good learning or debugging environment. From the application view point the hardware used for displaying of significant amounts of information that are to be scanned or read by a user is quite unsuitable. If no hard copy is required for the analysis of data base information the time sharing language involved must be capable of supporting CRT display units. The majority of such units are teletype code or ASCII code compatible. York APL does support such hardware devices and thus increases the number of applications that may be feasibly defined in APL. (Larger portions of the data base may be displayed to the user without pounding his ear drums or boring him by displaying information at 15 characters per second).

The last question leads us into the heart of the topic I wish to discuss. The access the language allows to existing data based information. In the majority of instances there is little or no conversion of data, required to access information on a standard O/S physical or indexed sequential file. The remainder of my presentation will be devoted to explaining briefly how these files are structured and how they are accessed by the York APL user.

PHYSICAL SEQUENTIAL FILES

The following is a brief description of fixed length blocked or unblocked sequential files. It should be noted that York APL does not allow the user access to physical sequential files of other structure. (See Figure 1)

For each file on a disk storage device there exists a name and a structure. The file may be described as a named set of vectors. A single member or vector of a named sequential file is normally called a record. The structural information is normally defined when the file is initially created via the batch O/S job stream. The vector containing this structural information is inaccessible to the APL user. It is technically named the Data Control Block (DCB).

When the user initially defines a file on a disk pack he may leave it empty or he may fill any portion of the file with specified data. When the user wishes to access a file stored on disk he must first set up a standard mode of communication between APL and the operating system under which APL executes. The user initially establishes this connection by executing the following function.

```
  ∇Z← N  OPEN  NAME
[1]Z← (N;NAME) ⌶7 ∇
  ∇Z← N  OPEN  NAME
[1]Z← (('o'$N);NAME)⌶7 ∇
```

to execute this function the user merely types:

```
1  OPEN  'LUCY'
```

where LUCY was the DSNAME created by an O/S batch programme.

At this point basically two things happen. First, APL verifies that the file does indeed exist and that the user may access the data set or file. Second, a file number is associated with or tied to the file name. In this case the number 1 is tied to the file named LUCY. From this point onward the number 1 is associated with the file when any input or output operations are performed.

Another thing that should be mentioned is that the name LUCY must be entered as the upper case characters of LUCY.

In other words ⎕↓∩↑. This translation can be done conveniently by executing a simple APL function.

Notice that the OPEN function has in reality been implemented as a primitive dyadic ibeam operator. For example a user could open the same file by executing the following APL statement

```
'1⎕↓∩↑'I7
```

In this case the function OPEN has been defined merely because its purpose is more easily remembered in that form. In fact all of the I/O operation that may be performed by York APL have been implemented as primitive dyadic ibeam functions. A list of some of these functions is shown below:

OPEN	(N;NAME) I7
CLOSE	(;N) I13
READ	(N;R) I12
WRITE	(N;R) I14
POINT	(N,M) I5

As you can see from the right hand column the number of the ibeam function has very little order and hence difficult to remember. The semi-colon used in the right column converts the number N to a character vector, in this case of length one.

Once the file is opened the user must communicate to APL information regarding the structure of the records in the file. This structure is more commonly referred to as the record format. The format information is generated by executing a primitive dyadic ibeam operator. The introduction of an example at this point will best illustrate the use of this I/O format operator.

Let us attempt to read a record from file 1.

```
X← 'A← 30C, B← 3R, C← 2P5'  I9
1  READ  'X, A, B, C'
```

The ibeam 9 operator assigns a vector of values to the variable X. This vector contains the format information used by the READ function to determine the lengths and data types of each of the variables listed in the READ function operand. In this case execution of the READ function will define three variables in the workspace. Variable A. a character vector of length 30, variable B a numeric vector of length 3 and variable C a numeric vector of length 2 are all defined from the information in the record retrieved from the file. The user is also allowed to be selective in the variables selected from the record. For example the execution of the statement

```
1  READ  'X,C'
```

would merely define the variable C to be a numeric vector of length 2 in the user's workspace. Any variables that are not explicitly used in the operand of the READ function are not defined or redefined in the users' workspace.

One should note that the format information defined by the vector X does not describe the format in which APL maintains the information in the users' workspace but only the format that the information has or is to have on the O/S file.

The various types of data supported by York APL are described in the table below:

Data Type Code	Result Shape	Data Type on Disk	Result Type	Disk Space in Bytes	Foot-Note
C	M	CHARACTER	CHARACTER	M	
D	M	D.P. Floating Point	Numeric	M X 8	
F	M	Fullword Fixed Point	Numeric	M X 4	
H	M	Half word Fixed Point	Numeric	M X 2	
P	M	Packed decimal	Numeric	*	*
R	M	S.P. Floating Point	Numeric	M X 4	
X	NA	NA	NA	M	**

* For packed decimal data the user must indicate the number of elements to be read and the length of each. (E.G. A← 2 P5 requests 2 packed decimal numbers each of length 5. to be assigned to A.)

** Data code X requests that vector or record elements be ignored. (E.G. 30X request 30 bytes of information to be skipped).

If the user has created his file so that the records are or will be stored on the disk in a fixed blocked structure, then each read performed will retrieve the next record from the file. The user may not generate a request to backspace one or more records on the file. He may only process the file information in a forward sequential manner. Writing records on a file is handled the same way as reads except that records are written rather than read.

If the user creates his file so that the records are not blocked, then he is allowed to retrieve any specified record by first pointing to it. For example if one executes the following, the read will retrieve the 3rd record from file one.

```
1  POINT  3
1  READ   'X,C'
```

It should be noted that in order to gain this type of power in the significantly more disk space than he would have used under a fixed blocked structure. Thus the user pays for added convenience in terms of added disk space. If the file is a source deck for another language and the user wished to use the file as input to a compiler, then the data may not be blocked and the POINT operation becomes extremely powerful. In this case the user may select any specified source card to be updated, without having to search the entire file.

I should now like to briefly switch the discussion to the topic of Indexed Sequential files.

INDEXED SEQUENTIAL FILES

The basic difference between the physical and indexed sequential file is that the index file has a key associated with each record or vector of data. The purpose of the key is to allow retrieval of file information by specifying a unique character string. For example if one has information associated with a car licence number on a file then this information may be retrieved by passing the licence number to the file system. This would be done by executing the following function:

```
1  KEY  'K88334'
1  READ 'X, A, B, C'
```

where the licence number is K88334 and the information requested is to be placed in the variables A, B and C. Notice that the READ function is the same as the one used for physical sequential files. The only difference here is that the key is specified before the read is performed.

When the user wishes to change a record on an ISAM file he must first read the record from the file. He may then use the RENRITE function to update that portion of the record defined in his workspace. For example if one wished to update the information in variable B mentioned earlier one would execute the following functions.

```
1  KEY  'K88334'
1  READ 'X, B'
B← 80ρ'*'
1  REWRITE 'X,B'
```

Note that it is not necessary to bring the entire record into the workspace but only that portion to be updated.

If new information is to be inserted into the file the user must supply the unique key for the record to be added. To write a record into an ISAM file one executes a WRITE function.

```
1  WRITE  'X, A, B, C'
```

where, as before X is the format and A, B and C are the data items. The key that is to be used to retrieve the record is imbedded in the record written. That is it is in a portion of A, B or C.

All newly created records of an ISAM file are placed in the overflow area of the file. Records may be flagged for deletion in either the overflow area or the prime area of the file in the normal way. In order to save valuable time on records in the overflow area to a minimum. This indicates that the normal maintenance of an ISAM file be done at reasonably frequent intervals.

One function that is supplied to the users for controlling error messages is called FMSG. This function controls the method of error message display for file functions. If this function has not been executed when an error occurs during execution of a file I/O Function, then a message is displayed on the console. For example if the end of file is searched the user might expect to have the following displayed:

```
Z←(N;R) I12 :[1] READ
          ? END OF DATA
```

However if the user has executed the FMSG function as follows:

FMSG 1 then when an error message is to be displayed for file 1 the user will receive a numeric 0 as a result.

For example:

```
        1  READ 'X,A'

0
```

indicates that an error occurred during the read function. If the read is successful the user normally receives a 1 as the result.

CONCLUSION

APL has not had the ability to access standard files for very long. However in this short period APL has been used in building data entry systems and information retrieval systems. It is currently being used for maintaining information for CAI courses in APL. In time other CAI courses in APL. In time other CAI material will make heavy use of data bases. Hopefully in this light APL will play a major role.

For researchers file I/O in APL opens up a whole new world. They may perform analysis on everything from data base structures, filing systems and queuing systems to analysis of current data that is on standard O/S files.

Above all the ability to access standard files via APL means that the workspace has been further expanded and that APL may now communicate with the outside world.

A P P E N D I X

REGISTRANTS FOR THE A.P.L. CONFERENCE
THURSDAY, JANUARY 27TH, 1972.

1. Mr. M. A. Griffiths,
   Director of Administrative Planning,
   Ryerson Polytechnical Institute.

2. Mr. A. Sauro,
   Dean of Applied Arts,
   Ryerson Polytechnical Institute.

3. Mr. G. Lavery,
   Instructor of Computer Science,
   Ryerson Polytechnical Institute.

4. Mr. A. Barclay,
   General Manager,
   Ryerson Polytechnical Institute.

5. Dr. H. H. Yates,
   Vice-President, Academic
   Ryerson Polytechnical Institute.

6. Mr. G. Collins,
   Consultant,
   Ryerson Polytechnical Institute.

7. Mr. J. Packham,
   Chairman, Electrical Technology Department,
   Ryerson Polytechnical Institute.

8. Mr. J. Abrahams,
   Director of Ryerson Systems Institute,
   Ryerson Polytechnical Institute.

9. Mr. S. Molder,
   Chairman of Mechnical Technology,
   Ryerson Polytechnical Institute.

10. Mr. I. A. Morgulis,
Associate Dean of Technology,
Ryerson Polytechnical Institute.

11. Mr. H. Zaplacinski,
Programmer Analyst,
Ryerson Polytechnical Institute.

12. Miss M. Yamanaka,
Programmer Analyst,
Ryerson Polytechnical Institute.

13. I. Taylor,
Manager of Academic Computer Services,
Ryerson Polytechnical Institute.

14. Mr. Larry Moore,
Director, Computer Centre,
Ryerson Polytechnical Institute.

15. Mr. J. Welland,
Manager of Administrative Computing Services,
Ryerson Polytechnical Institute.

16. Mr. L. Levitt,
Programmer Analyst,
Ryerson Polytechnical Institute.

17. Mr. Morgan Smyth,
A.P.L. Coordinator,
Ryerson Polytechnical Institute.

18. Mr. P. Barta,
Institute of Electrical Technology,
Ryerson Polytechnical Institute.

19. Mr. P. Stott,
Instructor of Computer Science,
Ryerson Polytechnical Institute.

20. Mr. A. Gilles,
Instructor of Mathematics,
Ryerson Polytechnical Institute.

21. Fred Balchunas,
Ryerson Polytechnical Institute.

22. J. MacLeod,
Ryerson Polytechnical Institute.

23. M. J. Hodgson,
Ryerson Polytechnical Institute.

24. Dr. Abdelmessih,
Ryerson Polytechnical Institute.

25. Dr. Marian B. Shepherd,
Assistant Professor, Computer Science,
York University.

26. Mr. Z. B. Dienes,
Manager, Audio Visual Department,
York University.

27. Mr. Hass Jamani,
Instructional Aid Resources,
York University.

28. Dr. D. E. Coates,
Data and Systems Analysis,
York University.

29. Mr. Dan Lamont,
M. B. A. Student,
York University.

30. Dr. Graham Haley,
Psychological Services,
York University.

31. Mr. John Tibert,
Institute of Behavioural Research,
York University.

32. Mr. Terry Mahoney,
Instutute of Behavioural Research,
York University.

33. Lynn Holmes,
Institute of Behavioural Research,
York University.

34. Mr. Ira Goldhar,
Institute of Behavioural Research,
York University.

35. Mirka Ondracek,
Institute of Behavioural Research,
York University.

36. Dr. Samuel Madras,
Director, Liberal Science Programme,
York University.

37. Mr. F. D. Simpkin,
Director, Computer Services,
York University.

38. Mr. B. H. Miller,
Assistant Director, Computer Services,
York University.

39. Mrs. J. I. Murphy,
Assistant Director, Computer Services,
York University.

40. Mr. H. P. Anderson,
Software Programmer,
York University.

41. Miss Irene Gates,
A.P.L. Programmer,
York University.

42. Mr. Don Genner,
Supervisor, Software Programming,
York University.

43. Mr. R. A. George,
Time Sharing Coordinator,
York University.

44. Mr. George Clapham,
Software Programmer,
York University.

45. Mr. D. I. Kellett,
Education Coordinator,
York University.

46. Mr. Chris Rickwood,
A.P.L. Programmer,
York University.

47. Miss Jane Takaoka,
Software Programmer,
York University.

48. Mr. Gerald E. Walker,
Assistant Professor,
York University.

49. Mr. David Ingram,
Lecturer,
York University.

50. Mrs. Eva I. Engelhardt,
Student,
Atkinson College.

51. Professor S. Book,
Faculty of Administrative Studies,
York University.

52. Professor R. Blackmore,
Faculty of Administrative Studies,
York University.

53. Professor J. Vasoff,
Faculty of Administrative Studies,
York University.

54. Professor U. Zohar,
Faculty of Administrative Studies,
York University.

55. Jim Mason,
Atkinson College,
York University.

56. F. E. Bunn,
Physics,
York University.

57. Sylvia Tewsley, Technical Writer,
York University.

58. Mr. Gene Romaniuk,
Associate Analyst,
IBM Canada Ltd.

59 Mr. John Pym,
APL Marketing Rep.,
I.P. Sharp A.

60. Valerie Chesters,
A.P.L. Specialist,
I.P. Sharp Assoc.

61. John Bassingthwaighte,
A.P.L.
I.P. Sharp Assoc.

62. George Horner,
Educational Rep.,
I.P. Sharp Assoc.

63. Ralph G. Morrison,
A.P.L. Counsellor,
I.P. Sharp Assoc.

64. Willie E. Laurila,
Programmer Analyst,
Canadian Development Division,
Control Data Canada Ltd.

65. Mr. Ron Seaberg,
Finance Manager,
Xerox of Canada Ltd.,

66. Mr. Gordon Ramer,
Director, Computer Centre,
St. Lawrence College of Applied Arts.

67. D. N. Inkster,
Faculty Computer Consultant,
Laurentian University.

68. J. C. Wilson,
Director, Computer Centre,
University of Toronto.

69. S. Goldfarb,
Manager,
University of Toronto.

70. Dr. Malcolm G. Lane,
Assistant Professor, Computer Science,
West Virginia University.

71. I. A. Davidson,
Associate Professor,
University of Waterloo.

72. Mary Gibson,
A.P.L. Coordinator,
University of Guelph.

73. Sharon Hayes,
A.P.L. Coordinator,
University of Guelph.

74. Mr. Maurice Lavimodiere,
Professor, Marketing Dept.,
Algonquin College.

75. Mrs. Gwen Williams,
University of Waterloo, A.P.L. Consultant,
University of Waterloo.

76. Professor E. H. Anthony,
Proffessor,
University of Guelph.

77. Mr. W. G. Harman,
Faculty Member,
Centennial College.

78. Mr. William R. Bezanson,
Lecturer,
Carleton University.

79. Mr. R. Swanson,
Faculty Member,
Humber College.

80. Mr. D. Cassel,
Faculty Member,
Humber College.

81. Mr. D. Watson,
Academic Advisor,
Lakehead University.

82. Mr. Les Oliver,
Data Processing,
Confederation College.

83. Mr. Benn,
Faculty Member,
Durham College.

84. Mr. John Cassidy,
Programmer Coordinator,
Niagara College.

85. Mr. Tom Honey,
Data Processing,
Niagara College.

86. Mr. Jacques Haguel,
Dept. of Math,
University of Sherbrooke.

87. Mr. Pierre Dion,
Dept. of Math,
University of Sherbrooke.

88. Mr. Jean Goulet,
Dept. of Math,
Sherbrooke University.

89. Mr. Bertrand Daigneault,
Dept. of Math,
University of Sherbrooke.

90. Jane Chaudhuri,
Dept. of Education,
University of Sherbrooke.